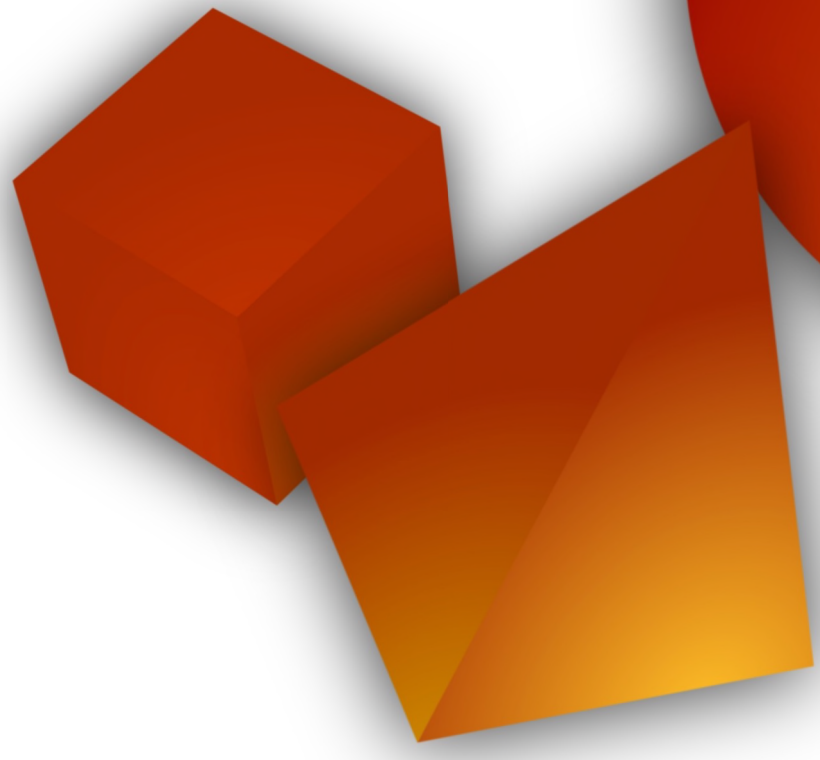
Referenzhandbuch

VERSION 2.5



Borland® ObjectWindows®

Borland C++ 4.5

ObjectWindows® Referenzhandbuch

BORLAND

Copyright © 1987, 1994 by Borland International, Inc.
Alle Rechte vorbehalten.

Borland hat möglicherweise Patente oder Patentanmeldungen für Teile des hier behandelten Themas. Dieses Handbuch gibt Ihnen keine Rechte auf diese Patente.

Alle Borland-Produkte sind Warenzeichen oder eingetragene Warenzeichen von Borland International, Inc. Andere aufgeführte Waren- und Produktbezeichnungen sind Warenzeichen oder eingetragene Warenzeichen der jeweiligen Rechteinhaber.

Einschränkung der Gewährleistung:
Es wird keine Garantie für die Richtigkeit des Inhaltes dieses Handbuches übernommen. Da sich Fehler, trotz aller Bemühungen, nie vollständig vermeiden lassen, sind wir für Hinweise jederzeit dankbar.

Herausgeber:

Borland GmbH
Monzastraße 4c
63225 Langen

Satz, Layout & Acrobat-Bearbeitung:

ALFES Computers, Publishing & Graphics, Siegen

Printed in Ireland.

Inhalt

Einführung 27
Inhalt dieses Handbuchs 27
Symbole und Typographie in diesem Handbuch 28
Konventionen in diesem Handbuch 29

Kapitel 1 ObjectWindows im Überblick 33
Hierarchiediagramm 33
Gebrauch der ObjectWindows Klassen . 36
Basisklassen 36
Klassen zur Fensterverwaltung 37
Rahmenfenster 37
Dekorierte Fenster 37
Allgemeine Dialoge 38
Steuerelemente 38
Gadgets 39
Menüs 39
Klassen zur Modulverwaltung 39
Klassen für das Dok/View-Modell 39
Drucker und Drucken 40
Klassen von Grafikobjekten 40
DC-Klassen 40
GDI-Klassen 41
Prüfungsklassen 41
Klassen zum Exception Handling 41
Befehlsaktivierungsklassen 42
ObjectSupport-Bibliotheksklassen 42
Die ObjectWindows-Bibliotheken 43
Compiler-Optionen zur Erstellung und Benutzung der ObjectWindows Bibliotheken. 44
Erstellen der ObjectWindows Bibliotheken 44
Verwendung der ObjectWindows Bibliothken 45
Die ObjectWindows-Header-Dateien 47
Die ObjectWindows-Ressourcendateien . . . 50
ObjectWindows Datentypen 51

Kapitel 2 ObjectWindows-Referenz . 53
Klasse TBird [Beispiel] 53
Typdefinitionen 53
Öffentliche Datenelemente 54
Öffentliche Konstruktoren und Destruktor 54
Öffentliche Elementfunktionen 54
Geschützte Datenelemente 55
Geschützte Konstruktoren 55
Geschützte Elementfunktionen 55
Antworttabelleneinträge 56
Schalter-Flag-Konstanten BF_xxxx 56
Schalterbotschaftskonstanten BF_xxxx . . . 56
Kombinationsfenster-Botschafts-Konstanten CBN_xxxx 57
Editkonstanten CM_xxxx 57
Editierdateikonstanten CM_xxxx 58
Bearbeiten-Ersetzen-Konstanten CM_xxxx 58
Edit View-Konstanten CM_xxxx 58
MDI-Konstanten CM_xxxx 59
Makro DECLARE_RESPONSE_TABLE . 59
Makro DEFINE_APP_DICTIONARY 60
Makro DEFINE_DOC_TEMPLATE_CLASS 60
Makros DEFINE_RESPONSE_TABLE . . . 60
Dialogelement-Botschaftskonstanten DLGC_xxxx 61
Dokumentverwaltungs-Modus-Konstanten dmxxxx 62
Dokumentbotschafts-Konstanten dnxxxx . . 62
Dokument-Template-Konstanten dtxxxx . . 62
Makro END_RESPONSE_TABLE 63
Editierbotschaftskonstanten EN_xxxx 64
Makros EV_xxxx 64
Funktion GetApplicationObject 65
Funktion GetWindowPtr 65

Dateikonstanten ID_xxxx 66
Druckerkonstanten ID_xxxx 66
Tastenkürzel-ID-Konstanten IDA_xxxx . . . 66
OLE-Tastenkürzel-ID-Konstanten IDA_xxxx 66
Menü-ID-Konstante IDM_xxxx 67
OLE-Menü-ID-Konstante IDM_xxxx 67
Edit-View-ID-Konstanten IDS_xxxx 67
Modus-Konstanten IDS_xxxx 67
Dokument-String-ID-Konstanten IDS_xxxx 68
Editierdatei-ID-Konstanten IDS_xxxx 68
Exception-Botschaften IDS_xxxx 68
List-View-ID-Konstanten IDS_xxxx 69
Drucker-String-ID-Konstanten IDS_xxxx . 70
Prüfungs-ID-Konstanten IDS_xxxx 70
Konstante IDW_MDICLIENT 70
Konstante IDW_MDIFIRSTCHILD 70
Klassen-Templates Factory 71
Factory-Klasse für nicht-automatisierte Dok/View-OLE-Komponenten . . . 71
Factory-Klasse für automatisierte Dok/View-OLE-Komponenten . . . 72
Factory-Klasse für nicht automatisierte Nicht-Dok/View-OLE-Komponenten 72
Factory-Klasse für automatisierte Nicht-Dok/View-OLE-Komponenten 72
ID-Konstanten LangXxxx 75
Listenfensterbotschafts-Konstanten LBN_xxxx 75
Konstante lmParent 76
Funktion LongMulDiv 76
Konstante MAX_RSRC_ERROR_STRING 76
Botschafts-Konstanten MB_Xxxx 76
Funktion NBits 77
Funktion NColors 78
Dokument-Öffnen-Aufzählung ofxxxx . . . 78
Eigenschaftsattributkonstanten pfxxxx 79
Makro _BUILDOWLDLL 79
Makro _OWLCLASS 79
Makro _OWLDATA 80
Makro _OWLDLL 80
Makro _OWLFAR 81
Makro _OWLFARVTABLE 81
Makro _OWLFASTTHIS 81
Makro _OWLFUNC 81
Funktion OWLGetVersion 82
Bildlaufleisten-Konstanten SB_Xxxx 82
Aufzählung shxxxx Dokument-Sharing . . . 82
Typdefinition TActionFunc 82
Typdefinition TActionMemFunc 83
Typdefinition TAnyPMF 83
Typdefinition TAnyDispatcher 83
Klasse TApplication 83
Öffentliche Datenelemente 83
Typdefinitionen 84
Öffentliche Konstruktoren und Destruktor . 85
Öffentliche Elementfunktionen 85
Geschützte Datenelemente 90
Geschützte Elementfunktionen 91
Klasse TApplication::TXInvalidMainWindow . . . 92
Öffentliche Konstruktoren 93
Öffentliche Elementfunktionen 93
Klasse TAppDictionary 93
Typdefinitionen 95
Öffentlicher Konstruktor und Destruktor . . 95
Öffentliche Elementfunktionen 96
Klasse TAutoFactory<> 97
Öffentliche Elementfunktionen 97
Klasse TBandInfo 98
Klasse TBitmap 99
Öffentliche Konstruktoren 99
Öffentliche Elementfunktionen 100
Geschützte Elementfunktionen 102
Klasse TBitmapGadget 102
Öffentliche Konstruktoren und Destruktor . 102
Öffentliche Elementfunktionen 102
Geschützte Elementfunktionen 102
Klasse TBitSet 103
Konstruktoren 103
Öffentliche Elementfunktionen 103
Klasse TBIVbxLibrary 104
Öffentlicher Konstruktor und Destruktor . . 105
Klasse TBrush 105

Öffentliche Konstruktoren 105
Öffentliche Elementfunktionen 106
Klasse TButton 106
Öffentliche Datenelemente 107
Öffentliche Konstruktoren 107
Geschützte Datenelemente 107
Geschützte Elementfunktionen 107
Antworttabelleneinträge 108
Klasse TButtonGadget 108
Typdefinitionen 109
Öffentliche Konstruktoren und Destruktor 109
Öffentliche Elementfunktionen 110
Geschützte Datenelemente 111
Geschützte Elementfunktionen 112
Klasse TButtonGadgetEnabler 114
Öffentlicher Konstruktor 114
Geschütztes Datenelement 114
Öffentliche Elementfunktionen 115
Klasse TCelArray 115
Öffentliche Konstruktoren und Destruktor 115
Öffentliche Elementfunktionen 116
Geschützte Datenelemente 117
Klasse TCharSet 117
Öffentliche Konstruktoren 117
Öffentliche Elementfunktionen 118
Klasse TCheckBox 118
Öffentliche Datenelemente 118
Öffentliche Konstruktoren 118
Öffentliche Elementfunktionen 119
Geschützte Elementfunktionen 120
Antworttabelleneinträge 121
Klasse TChooseColorDialog 121
Öffentliche Konstruktoren 121
Öffentliche Elementfunktionen 122
Öffentliche Datenelemente 122
Geschützte Elementfunktionen 122
Antworttabelleneinträge 123
Klasse TChooseColorDialog::TData 123
Öffentliche Datenelemente 123
Klasse TChooseFontDialog 124
Öffentliche Konstruktoren 124
Geschützte Datenelemente 124
Geschützte Elementfunktionen 124
Antworttabelleneinträge 125
Klasse TChooseFontDialog::TData 125
Öffentliche Datenelemente 125
Klasse TClientDC 127
Öffentliche Konstruktoren 127
Klasse TClipboard 127
Öffentlicher Destruktor 129
Öffentliche Datenelemente 129
Öffentliche Elementfunktionen 129
Geschützte Datenelemente 133
Geschützter Konstruktor 133
Klasse TClipboardViewer 133
Geschützte Datenelemente 133
Geschützte Konstruktoren 133
Geschützte Elementfunktionen 134
Antworttabelleneinträge 134
Klasse TColor 135
Öffentliche Datenelemente 135
Öffentliche Konstruktoren 136
Öffentliche Elementfunktionen 137
Geschützte Datenelemente 138
Klasse TComboBox 139
Öffentliche Konstruktoren 139
Öffentliche Datenelemente 139
Öffentliche Elementfunktionen 140
Geschützte Elementfunktionen . . . 143
Klasse TComboBoxData 144
Öffentliche Konstruktoren und Destruktor 144
Öffentliche Elementfunktionen 144
Geschützte Datenelemente 145
Klasse TCommandEnabler 146
Behandlung von Befehlsbotschaften 146
Reaktion auf Befehlsbotschaften 146
Befehlsverarbeitung in TWindow 147
Befehlsverarbeitung in TFrameWindow . . . 147
EvCommand aufrufen 147
Verarbeitung von Befehlsaktivierungsbotschaften 148
Befehlsaktivierung für Menüeinträge 148
Befehlsaktivierung für Mauspalettenschalter 148
Erzeugen spezialisierter Befehlsaktivierer . 148
Verwendung des Makros EV_COMMAND_ENABLE 148

Automatisches Aktivieren und Deaktivieren von Befehlen 149
Öffentliche Konstruktoren 149
Typdefinitionen 149
Öffentliche Datenelemente 150
Öffentliche Elementfunktionen 150
Geschützte Datenelemente 150
Klasse TCommonDialog 150
Öffentliche Konstruktoren 151
Öffentliche Elementfunktionen 152
Geschützte Datenelemente 152
Geschützte Elementfunktionen 152
Antworttabelleneinträge 153
Typdefinition TCondFunc 153
Typdefinition TCondMemFunc 153
Klasse TControlBar 153
Öffentliche Konstruktoren 154
Öffentliche Elementfunktionen 154
Geschützte Elementfunktionen 154
Klasse TControlGadget 155
Öffentliche Konstruktoren und Destruktor 155
Geschützte Datenelemente 155
Geschützte Elementfunktionen 155
Antworttabelleneinträge 156
Klasse TControl 156
Öffentliche Konstruktoren 157
Öffentliche Elementfunktionen 157
Geschützte Elementfunktionen 158
Antworttabelleneinträge 159
Klasse TCreatedDC 159
Öffentliche Konstruktoren und Destruktor . 159
Geschützte Elementfunktionen 160
Klasse TCursor 160
Öffentliche Konstruktoren und Destruktor . 160
Öffentliche Elementfunktionen 161
Operatoren 161
Klasse TDC 161
Öffentliche Konstruktoren und Destruktor 162
Öffentliche Elementfunktionen 162
Geschützte Konstruktoren 207
Geschützte Datenelemente 207
Geschützte Elementfunktionen 208
Klasse TDecoratedFrame 209
Typdefinitionen 209
Öffentlicher Konstruktor 209
Öffentliche Elementfunktionen 209
Geschützte Datenelemente 210
Geschützte Elementfunktionen 210
Anworttabelleneinträge 211
Klasse TDecoratedMDIFrame 211
Öffentliche Konstruktoren 211
Geschützte Elementfunktion 212
Anworttabelleneinträge 212
Klasse TDesktopDC 212
Öffentlicher Konstruktor 212
Klasse TDialog 212
Öffentliche Datenelemente 213
Öffentliche Konstruktoren und Destruktor . 213
Öffentliche Elementfunktionen 214
Geschützte Elementfunktionen 217
Anworttabelleneinträge 218
Klasse TDialog 218
Öffentliche Datenelemente 219
Klasse TDib 219
Öffentliche Konstruktoren und Destruktor . 220
Öffentliche Elementfunktionen 221
Geschützte Datenelemente 225
Geschützte Elementfunktionen 226
Klasse TDibDC 226
Öffentliche Konstruktoren 226
Klasse TDocument 226
Öffentliche Datenelemente 227
Typdefinition 228
Öffentliche Konstruktoren und Destruktor . 228
Öffentliche Elementfunktionen 228
Geschützte Datenelemente 234
Geschützte Elementfunktionen 234
Eingebettete Klasse TDocument::List 234
Öffentliche Konstruktoren und Destruktor . 234
Öffentliche Elementfunktionen 234
Klasse TDocManager 235
Öffentliche Konstruktoren und Destruktor . 236
Öffentliche Datenelemente 237
Öffentliche Elementfunktionen 237
Geschützte Elementfunktionen 241
Anworttabelleneinträge 242
Struktur TEdgeConstraint 242
Öffentliche Elementfunktionen 243
Struktur TEdgeOrSizeConstraint 244
Öffentliche Elementfunktionen 244

Klasse TEdit 245
Öffentliche Konstruktoren 246
Öffentliche Elementfunktionen 246
Geschütztes Datenelement 252
Geschützte Elementfunktionen 252
Anworttabelleneinträge 254
Klasse TEditFile 255
Öffentliche Konstruktoren und Destruktor 255
Öffentliche Datenelemente 255
Öffentliche Elementfunktionen 255
Geschützte Elementfunktionen 257
Anworttabelleneinträge 257
Klasse TEditSearch 258
Öffentliche Konstruktoren 258
Öffentliche Datenelemente 258
Öffentliche Elementfunktionen 258
Anworttabelleneinträge 259
Klasse TEditView 259
Öffentliche Konstruktoren und Destruktor 259
Öffentliche Elementfunktionen 260
Geschützte Datenelemente 261
Geschützte Elementfunktionen 261
Anworttabelleneinträge 262
Typdefinition TEqualOperator 262
Klasse TEventHandler::TEventInfo 262
Öffentliche Konstruktoren 263
Aufzählungstyp TEventStatus 263
Öffentliche Datenelemente 263
Klasse TEventHandler 264
Öffentliche Elementfunktionen 264
Geschützte Elementfunktionen 264
Klasse TFileDocument 265
Öffentliche Konstruktoren und Destruktor 265
Typdefinition 265
Öffentliche Elementfunktionen 266
Geschützte Datenelemente 268
Geschützte Elementfunktionen 268
Klasse TFileOpenDialog 268
Öffentliche Konstruktoren 268
Öffentliche Elementfunktionen 269
Klasse TFileSaveDialog 269
Öffentliche Konstruktoren 269
Öffentliche Elementfunktionen 269
Klasse TFilterValidator 269
Öffentliche Konstruktoren 270
Öffentliche Elementfunktionen 270
Geschützte Datenelemente 270
Klasse TFindDialog 270
Öffentliche Konstruktoren 271
Geschützte Elementfunktionen 271
Klasse TFindReplaceDialog 271
Öffentliche Konstruktoren 271
Öffentliche Elementfunktionen 272
Geschützte Datenelemente 272
Geschützte Elementfunktionen 272
Anworttabelleneinträge 273
Klasse TFindReplaceDialog::TData 273
Öffentlicher Konstruktor und Destruktor 273
Öffentliche Datenelemente 273
Klasse TFloatingFrame 275
Öffentliche Konstruktoren 275
Öffentliche Elementfunktionen 276
Geschützte Elementfunktionen 276
Anworttabelleneinträge 276
Klasse TFont 276
Öffentliche Konstruktoren 276
Öffentliche Elementfunktionen 277
Klasse TFrameWindow 278
Öffentliche Konstruktoren und Destruktor 278
Öffentliche Datenelemente 279
Öffentliche Elementfunktionen 279
Geschützte Datenelemente 282
Geschützter Konstruktor 282
Geschützte Elementfunktionen 282
Anworttabelleneinträge 284
Klasse TGadget 284
Öffentliche Konstruktoren und Destruktor 285
Öffentliche Datenelemente 285
Öffentliche Aufzählungstypen und Strukturen 285
Öffentliche Elementfunktionen 286
Geschützte Datenelemente 288
Geschützte Elementfunktionen 289
Klasse TGadgetWindowFont 291

Öffentlicher Konstruktor 292
Klasse TGadgetWindow 292
Öffentlicher Konstruktor und Destruktor 293
Typdefinitionen 293
Öffentliche Elementfunktionen 293
Geschützte Datenelemente 296
Geschützte Elementfunktionen 297
Antworttabelleneinträge 299
Klasse TGauge 300
Öffentliche Konstruktor 300
Öffentliche Elementfunktionen 300
Geschützte Datenelemente 301
Geschützte Elementfunktionen 301
Antworttabelleneinträge 302
Klasse TGdiObject 302
Öffentlicher Destruktor 303
Typdefinitionen 303
Öffentliche Elementfunktionen 304
Geschützte Datenelemente 305
Geschützte Elementfunktionen 305
Geschützte Konstruktoren 306
Makros 306
Klasse TGdiObject::TXGdi 307
Öffentliche Konstruktoren 308
Öffentliche Elementfunktionen 308
Klasse TGroupBox 308
Öffentliche Datenelemente 309
Öffentliche Konstruktoren 309
Öffentliche Elementfunktionen 310
Klasse THatch8x8Brush 310
Öffentliche Datenelemente 311
Öffentlicher Konstruktor 312
Öffentliche Elementfunktion 312
Klasse TIC 312
Öffentliche Konstruktoren 312
Klasse TIcon 313
Öffentliche Konstruktoren und Destruktor 313
Öffentliche Elementfunktionen 313
Klasse TInputDialog 314
Öffentliche Datenelemente 314
Öffentliche Konstruktoren 314
Öffentliche Elementfunktion 315
Geschützte Elementfunktion 315
Klasse TInStream 315
Öffentliche Konstruktoren 315
Struktur TLayoutConstraint 315
Öffentliche Datenelemente 316
Klasse TLayoutMetrics 317
Öffentliche Datenelemente 318
Öffentliche Konstruktoren 318
Klasse TLayoutWindow 320
Beispiele 320
Öffentliche Konstruktoren und Destruktor 322
Öffentliche Elementfunktionen 322
Geschützte Datenelemente 323
Geschützte Elementfunktionen 323
Antworttabelleneinträge 323
Klasse TListBox 323
Öffentliche Konstruktoren 323
Öffentliche Elementfunktionen 324
Geschützte Elementfunktionen 329
Struktur TListBoxData 330
Öffentliche Datenelemente 330
Öffentliche Konstruktoren und Destruktor . 330
Öffentliche Elementfunktionen 330
Klasse TListView 332
Öffentliche Konstruktoren und Destruktor 332
Öffentliches Datenelement 332
Öffentliche Elementfunktionen 332
Geschützte Datenelemente 333
Geschützte Elementfunktionen 333
Antworttabelleneinträge 336
Struktur TLocaleString 336
Öffentliche Elementfunktionen 337
Klasse TLookupValidator 338
Öffentliche Konstruktoren 338
Öffentliche Elementfunktionen 338
Klasse TMDIChild 339
Öffentliche Konstruktoren und Destruktor 339
Öffentliche Elementfunktionen 340
Geschützte Elementfunktionen 340
Antworttabelleneinträge 341
Klasse TMDIClient 341
Öffentliche Konstruktoren und Destruktor 341

Öffentliches Datenelement 342
Öffentliche Elementfunktionen 342
Geschützte Elementfunktionen 343
Antworttabelleneinträge 345
Klasse TMDIFrame 345
Öffentliche Konstruktoren 346
Öffentliche Elementfunktionen 346
Geschützte Elementfunktion 347
Antworttabelleneinträge 347
Aufzählungstyp TMeasurementUnits 348
Klasse TMemoryDC 348
Öffentliche Konstruktoren 348
Öffentliche Elementfunktionen 348
Geschütztes Datenelement 349
Klasse TMenu 349
Öffentliche Konstruktoren und Destruktor 349
Öffentliche Elementfunktionen 350
Geschützte Datenelemente 354
Geschützte Elementfunktionen 354
Klasse TMenuDescr 355
Öffentliche Konstruktoren und Destruktor 358
Typdefinitionen 359
Öffentliche Elementfunktionen 359
Geschützte Datenelemente 360
Geschützte Elementfunktionen 360
Klasse TMenuItemEnabler 361
Öffentlicher Konstruktor 361
Geschützte Datenelemente 361
Öffentliche Elementfunktionen 361
Klasse TMessageBar 362
Öffentlicher Konstruktor 362
Öffentliche Elementfunktionen 362
Geschützte Datenelemente 362
Geschützte Elementfunktionen 363
Klasse TMetaFileDC 363
Konstruktoren und Destruktor 363
Öffentliche Elementfunktion 364
Klasse TMetaFilePict 364
Öffentliche Konstruktoren und Destruktor 364
Öffentliche Elementfunktionen 365
Geschützte Datenelemente 366
Klasse TModule 366
Öffentliche Konstruktoren und Destruktor 366
Öffentliche Datenelemente 367
Öffentliche Elementfunktionen 368
Geschützte Datenelemente 373
Klasse TModule::TxInvalidModule 373
Öffentliche Konstruktoren 373
Öffentliche Elementfunktionen 374
Klasse TOleClientDC 374
Öffentlicher Konstruktor 374
Klasse TOleDocument 375
Öffentlicher Konstruktor und Destruktor 375
Öffentliche Elementfunktionen 375
Klasse TOleFactoryBase<> 378
Öffentliche Elementfunktionen> Klasse;öffentliche Elementfunktionen> 379
Template-Argumente> Klasse;Template> 379
Klasse TOleFrame 380
Öffentlicher Konstruktor und Destruktor 381
Öffentliche Elementfunktionen 381
Geschützte Elementfunktionen 382
Geschützte Datenelemente 385
Einträge in der Anworttabelle 385
Klasse TOleMDIFrame 386
Öffentlicher Konstruktor und Destruktor 387
Geschützte Elementfunktionen 387
Einträge in der Anworttabelle 388
Klasse TOleView 388
Öffentliche Konstruktoren und Destruktoren 389
Öffentliche Elementfunktionen 389
Geschützte Elementfunktionen 389
Einträge in die Anworttabelle 391
Klasse TOleWindow 392
Öffentlicher Konstruktor und Destruktor 393
Öffentliche Elementfunktionen 393
Geschützte Datenelemente 394
Geschützte Elementfunktionen 396
Einträge in die Anworttabelle 407
Klasse TOpenSaveDialog 408

Öffentlicher Konstruktor 409
Öffentliche Elementfunktionen 409
Geschützte Datenelemente 409
Geschützte Konstruktoren 409
Geschützte Elementfunktionen 410
Antworttabelleneinträge 411
Struktur TOpenSaveDialog::TData 411
Öffentliche Konstruktoren und Destruktor 411
Öffentliche Datenelemente 411
Öffentliche Elementfunktionen 413
Klasse TOutStream 413
Öffentlicher Konstruktor 413
Klasse TPaint DC 413
Öffentliche Konstruktoren und Destruktor 414
Öffentliche Datenelemente 414
Klasse TPalette 414
Öffentliche Konstruktoren 414
Öffentliche Elementfunktionen 415
Geschützte Elementfunktionen 417
Klasse TPaletteEntry 418
Öffentliche Konstruktoren 418
Klasse TPen 419
Öffentliche Datenelemente 419
Öffentliche Konstruktoren 419
Öffentliche Elementfunktionen 420
Aufzählungstyp TPicResult 421
Aufzählungstyp TPlacement 421
Klasse TPopupMenu 421
Öffentliche Konstruktoren 421
Öffentliche Elementfunktionen 422
Klasse TPreviewPage 422
Öffentliche Konstruktoren 422
Öffentliche Elementfunktionen 423
Geschützte Datenelemente 423
Geschützte Elementfunktionen 423
Antworttabelleneinträge 424
Klasse TPrintDC 424
Öffentliche Konstruktoren 424
Öffentliche Elementfunktionen 424
Geschütztes Datenelement 434
Struktur TPrintDialog::TData 434
Öffentliche Datenelemente 435
Öffentliche Elementfunktionen 436
Klasse TPrintDialog 437
Öffentlicher Konstruktor 438
Öffentliche Elementfunktionen 438
Geschützte Datenelemente 438
Geschützte Elementfunktionen 439
Antworttabelleneinträge 439
Klasse TPrinter 439
Öffentlicher Konstruktor und Destruktor 439
Öffentliche Elementfunktionen 439
Geschützte Datenelemente 440
Geschützte Elementfunktionen 440
Klasse TPrinter::TXPrinter 441
Öffentlicher Konstruktor 441
Klasse TPrinterAbortDlg 441
Öffentliche Konstruktoren 442
Öffentliche Elementfunktionen 442
Klasse TPrintout 442
Öffentlicher Konstruktor und Destruktor 442
Öffentliche Elementfunktionen 443
Typdefinition 444
Geschützte Datenelemente 445
Klasse TPrintPreviewDC 445
Öffentlicher Konstruktor und Destruktor 446
Öffentliche Elementfunktionen 446
Geschützte Datenelemente 449
Geschützte Elementfunktionen 449
Klasse TProfile 450
Öffentlicher Konstruktor und Destruktor 450
Öffentliche Elementfunktionen 450
Klasse TPXPictureValidator 451
Öffentliche Konstruktoren 451
Öffentliche Elementfunktionen 451
Geschütztes Datenelement 452
Klasse TRadioButton 453
Öffentliche Konstruktoren 453
Geschützte Elementfunktionen 453
Antworttabelleneinträge 454
Klasse TRangeValidator 454
Öffentliche Konstruktoren 454
Öffentliche Elementfunktionen 454
Geschützte Datenelemente 455

Klasse TRegion 455
Öffentliche Datenelemente 455
Öffentliche Konstruktoren 455
Öffentliche Elementfunktionen 457
Aufzählung TRelationship 459
Klasse TReplaceDialog 459
Öffentliche Konstruktoren 459
Geschützte Elementfunktionen 459
Klasse TResponseTableEntry 460
Öffentliche Datenelemente 460
Klasse TRgbQuad 461
Öffentliche Konstruktoren 461
Klasse TRgbTriple 461
Öffentliche Konstruktoren 462
Klasse TScreenDC 462
Öffentliche Konstruktoren 463
Klasse TScrollBar 463
Öffentliche Datenelemente 463
Öffentliche Konstruktoren 463
Öffentliche Elementfunktionen 464
Geschützte Elementfunktionen 466
Antworttabelleneinträge 467
Struktur TScrollBarData 467
Öffentliche Datenelemente 467
Klasse TScroller 468
Öffentliche Datenelemente 468
Öffentliche Konstruktoren und Destruktor 469
Öffentliche Elementfunktionen 469
Klasse TSeparatorGadget 471
Öffentliche Elementfunktionen 472
Klasse TSlider 472
Öffentliche Konstruktoren und Destruktor 472
Öffentliche Elementfunktionen 473
Geschützte Elementfunktionen 473
Geschützte Datenelemente 477
Antworttabellen 478
Klasse THSlider 479
Öffentliche Konstruktoren 479
Geschützte Elementfunktionen 479
Klasse TVSlider 480
Öffentliche Konstruktoren 480
Geschützte Elementfunktionen 480
Klasse TSortedStringArray 481
Öffentlicher Konstruktor 481
Typdefinitionen 481
Öffentliche Elementfunktionen 482
Klasse TStatic 484
Öffentliche Datenelemente 484
Öffentliche Konstruktoren 484
Öffentliche Elementfunktionen 485
Geschützte Elementfunktionen 485
Antworttabelleneinträge 486
Klasse TStatus 486
Öffentliche Konstruktoren 486
Öffentliche Datenelemente 486
Klasse TStatusBar 486
Typdefinition 487
Öffentliche Konstruktoren 487
Öffentliche Elementfunktionen 488
Geschützte Datenelemente 489
Geschützte Elementfunktionen 489
Klasse TStorageDocument 490
Typdefinitionen 490
Öffentlicher Konstruktor und Destruktor 491
Öffentliche Elementfunktionen 491
Geschützte Datenelemente 494
Klasse TStream 494
Öffentlicher Destruktor 494
Öffentliche Elementfunktionen 495
Geschützte Datenelemente 495
Geschützte Konstruktoren 495
Klasse TStringLookupValidator 495
Öffentliche Konstruktoren und Destruktor 496
Öffentliche Elementfunktionen 496
Geschütztes Datenelement 496
Klasse TSystemMenu 496
Öffentliche Konstruktoren 497
Klasse TTextGadget 497
Öffentliche Datenelemente 497
Öffentliche Konstruktoren und Destruktor 497
Öffentliche Elementfunktionen 497
Geschützte Datenelemente 498
Geschützte Elementfunktionen 498
Typdefinition TTileDirection 499
Klasse TTinyCaption 499

Geschützte Datenelemente 500
Geschützte Konstruktoren und Destruktor 501
Geschützte Elementfunktionen 501
Antworttabelleneinträge 505
Klasse TToolBox 506
Öffentliche Konstruktoren 506
Öffentliche Elementfunktionen 506
Geschützte Datenelemente 506
Geschützte Elementfunktionen 507
Aufzählung TTransferDirection 507
Klasse TUIHandle 507
Typdefinitionen 508
Öffentlicher Konstruktor 509
Öffentliche Elementfunktionen 510
Klasse TValidator 511
Öffentliche Konstruktoren und Destruktor 511
Öffentliche Elementfunktionen 511
Geschützte Datenelemente 513
Klasse TValidator::TXValidator 514
Öffentliche Konstruktoren 514
Klasse TVbxControl 514
Öffentliche Konstruktoren und Destruktor 515
Öffentliche Elementfunktionen 515
Geschützte Elementfunktionen 520
Antworttabelleneinträge 520
Klasse TVbxEventHandler 521
Geschützte Elementfunktionen 525
Antworttabelleneinträge 525
Klasse TView 525
Öffentliche Datenelemente 526
Öffentliche Konstruktoren und Destruktor 526
Öffentliche Elementfunktionen 526
Geschützte Datenelemente 528
Geschützte Elementfunktionen 528
Aufzählung TWidthHeight 528
Struktur TWindowAttr 529
Öffentliche Datenelemente 529
Klasse TWindow 530
Öffentliche Datenelemente 531
Öffentliche Konstruktoren und Destruktor 532
Öffentliche Elementfunktionen 533
Geschützte Datenelemente 568
Geschützter Konstruktor 569
Geschützte Elementfunktionen 569
Antworttabelleneinträge 573
Klasse TWindowDC 574
Öffentliche Konstruktoren und Destruktor 574
Geschützte Konstruktoren 574
Geschütztes Datenelement 574
Klasse TWindowView 574
Öffentliche Elementfunktionen 575
Antworttabelleneinträge 575
Klasse TWindow::TXWindow 576
Öffentliche Konstruktoren 576
Öffentliche Datenelemente 576
Öffentliche Elementfunktionen 576
Klasse TXCompatibility 577
Öffentliche Konstruktoren 577
Öffentliche Elementfunktionen 577
Klasse TXMenu 577
Öffentliche Konstruktoren 577
Öffentliche Elementfunktionen 577
Klasse TXOutOfMemory 577
Öffentliche Konstruktoren 578
Öffentliche Elementfunktionen 578
Klasse TXOwl 578
Öffentliche Konstruktoren und Destruktor 580
Öffentliche Elementfunktionen 580
vnxxxx View-Benachrichtigungskonstanten 581

Kapitel 3
Die Ereignisbehandlungsroutinen von ObjectWindows 583

Kapitel 4
ObjectWindows-Zuteilungsfunktionen 591

Liste der Zuteilungsfunktionen von ObjectWindows 593
i_LPARAM_Dispatch 593
i_U_W_U_Dispatch 593
i_WPARAM_Dispatch 593

I32_Dispatch 593
I32_LPARAM_Dispatch 593
I32_WPARAM_LPARAM_Dispatch 593
I32_MenuChar_Dispatch 594
I32_U_Dispatch 594
U_Dispatch 594
U_LPARAM_Dispatch 594
U_POINT_Dispatch 594
U_POINTER_Dispatch 594
U_U_Dispatch 595
U_U_U_U_Dispatch 595
U_WPARAM_LPARAM_Dispatch . 595
v_Activate_Dispatch 595
v_Dispatch 595
v_LPARAM_Dispatch 595
v_MdiActivate_Dispatch 596
v_ParentNotify_Dispatch 596
v_POINT_Dispatch 596
v_POINTER_Dispatch 596
v_U_Dispatch 596
v_U_POINT_Dispatch 597
v_U_U_Dispatch 597
v_U_U_U_Dispatch 597
v_U_U_W_Dispatch 597
v_WPARAM_Dispatch 597
v_WPARAM_LPARAM_Dispatch . 598

Kapitel 5
Übersicht über ObjectComponents . 599

ObjectComponents-Bibliotheken 599
ObjectComponents-Header-Dateien 600
Allgemeine Klassen, Makros und Typdefinitionen für OLE 600
Globale Zusatzfunktionen 601
ObjectComponents-Exception-Klassen 601
Automatisierungsklassen 602
Aufzählungstypen und Typdefinitionen für Automatisierung 602
Automatisierungsdatentypen 602
Deklarationen/Definitionen von Automatisierungsdatentypen 603
Automatisierungsdeklarationsmakros 604
Deklarationsmakros 605
Makro AUTODETACH 605
Automatisierungsdefinitionsmakros 605
Hook-Makros für die Automatisierung 606
Beispiel 607
Platzhalter-Makros für die Automatisierung 607
Registrierschlüssel 608
Klassen für Verknüpfung und Einbettung . . 610
Aufzählungstypen für Verknüpfung und Einbettung 611
Botschaften für Verknüpfung und Einbettung 611
Strukturen für Verknüpfung und Einbettung 612
ocrxxxx-Konstanten 612

Kapitel 6
Bibliotheksreferenz ObjectComponents 613

Makro _ICLASS 613
Makro _IFUNC 613
Makros _OCFxxxx 613
Registrierschlüssel aspectall 614
Registrierschlüssel aspectcontent 614
Registrierschlüssel aspectdocprint 615
Registrierschlüssel aspecticon 615
Registrierschlüssel aspectthumbnail 616
Makros AUTOARGS 616
Makros AUTOCALL_xxxx 617
Makro _AUTOCLASS 618
Makros AUTODATA 618
Aufzählungstyp AutoDataType 619
Makro AUTODETACH 620
Makros AUTOENUM 620
Makro AUTOFLAG 621
Makros AUTOFUNC 621
Makro AUTOINVOKE 623
Makros AUTOITERATOR 623
Makros AUTONAMES 624
Makro AUTONOHOOK 625
Makros AUTOPROP 625
Makro AUTORECORD 626
Makro AUTOREPORT 627
Aufzählungstyp AutoSymFlag 627
Makros AUTOSTAT 628
Makro AUTOTHIS 629
Makro AUTOUNDO 629

Makro AUTOVALIDATE 630
Registrierschlüssel clsid 630
Registrierschlüssel cmdline 631
Registrierschlüssel debugclsid 631
Registrierschlüssel debugdesc 632
Registrierschlüssel debugger 632
Registrierschlüssel debugprogid 633
Makro DECLARE_AUTOCLASS 633
Makros DECLARE_COMBASES*n* 634
Makro DEFINE_AUTOAGGREGATE . . . 635
Makro DEFINE_AUTOCLASS 635
Makros DEFINE_COMBASES*n* 635
Registrierschlüssel description 636
Registrierschlüssel directory 637
Registrierschlüssel docfilter 637
Registrierschlüssel docflags 637
Funktion DynamicCast 638
Makro END_AUTOAGGREGATE 638
Makro END_AUTOCLASS 639
Makro EXPOSE_APPLICATION 639
Makro EXPOSE_DELEGATE 640
Makro EXPOSE_INHERIT 640
Makro EXPOSE_ITERATOR 641
Makros EXPOSE_METHOD 642
Makros EXPOSE_PROPxxxx 643
Makro 644
Registrierschlüssel extension 645
Registrierschlüssel filefmt 646
Registrierschlüssel format*n* 646
Registrierschlüssel handler 647
Registrierschlüssel helpdir 647
Rückgabekonstanten HR_xxxx 648
Registrierschlüssel iconindex 648
Registrierschlüssel insertable 649
Registrierschlüssel language 649
Registrierschlüssel menuname 649
Funktion MostDerived 650
Typdefinition ObjectPtr 650
OC_APPxxxx-Botschaften 650
OC_VIEWxxxx-Botschaften 651
ocrxxxx-Aspekt-Konstanten 652
ocrxxxx-Zwischenablage-Konstanten 653
ocrxxxx-Richtungskonstanten 654
ocrxxxx-Begrenzungskonstanten 654
ocrxxxx-Medienkonstanten 654
ocrxxxx-Objektstatuskonstanten 655
ocrxxxx-Verwendungskonstanten 656
ocrxxxx-Aktionsart-Attributkonstanten . . . 656
ocrxxxx-Aktionsartmenü-Flags 656
Makro OPTIONAL_ARG 657
Registrierschlüssel path 657
Registrierschlüssel permid 658
Registrierschlüssel permname 658
Registrierschlüssel progid 658
Makro REQUIRED_ARG 659
Klasse TAutoBase 659
Öffentlicher Destruktor 660
Destruktor 660
Struktur TAutoBool 660
Öffentliches Datenelement 660
ClassInfo 660
Klasse TAutoCommand 660
Öffentlicher Konstruktor und Destruktor 661
Konstruktor 661
Destruktor 661
Typdefinitionen 661
TCommandHook 661
TErrorMsgHook 661
Öffentliche Elementfunktionen 661
ClearFlag 661
Execute 662
Fail 662
GetSymbol 662
Invoke 662
IsPropSet 662
LookupError 662
Record 663
Report 663
Return 663
SetCommandHook 663
SetErrorMesgHook 663
SetFlag 664
SetSymbol 664
TestFlag 664
Undo 664
Validate 664
Geschützte Datenelemente 665
Attr 665
Symbol 665
Struktur TAutoCurrency 665

Öffentliches Datenelement 665
ClassInfo 665
Struktur TAutoDate 665
Öffentliche Datenelemente 665
ClassInfo 665
Date 666
Öffentliche Konstruktoren 666
Konstruktoren 666
Öffentliche Elementfunktion 666
operator double 666
Struktur TAutoDouble 666
Öffentliches Datenelement 666
ClassInfoClassInfo 666
Klasse TAutoEnumerator<> 666
Öffentliche Konstruktoren und Destruktor 667
Konstruktoren> 667
Destruktor> 667
Öffentliche Elementfunktionen 667
Bind 667
Clear> 668
Object> 668
Step> 668
Unbind> 668
Value> 668
Struktur TAutoFloat 669
Öffentliches Datenelement 669
ClassInfo 669
Klasse TAutoIterator 669
Öffentliche Elementfunktionen 670
Copy 670
GetSymbol 670
Init 670
IUnknown 671
Return 671
SetSymbol 671
Step 672
Test 672
Geschützte Konstruktoren 672
Konstruktoren 672
Geschütztes Datenelement 673
Owner 673
Struktur TAutoLong 673
Öffentliches Datenelement 673
ClassInfo 673
Klasse TAutoObject<> 673
Öffentliche Konstruktoren 673
Konstruktoren 673
Öffentliche Elementfunktionen 674
operator *() 674
operator = ... 674
operator T& () 674
operator T* () 674
TObjectDescriptor() 675
Geschütztes Datenelement 675
operator P 675
Klasse TAutoObjectByVal<> 675
Basisklasse 675
Öffentliches Datenelement 675
operator = ... 675
Öffentliche Konstruktoren 676
Konstruktoren 676
Klasse TAutoObjectDelete<> 676
Basisklasse> 676
Öffentliche Konstruktoren 676
Konstruktoren 676
Öffentliche Elementfunktionen 677
operator = ... 677
TObjectDescriptor() 677
Klasse TAutoProxy 677
Öffentlicher Destruktor 678
Destruktor 678
Öffentliche Elementfunktionen 678
Bind 678
IDispatch&() 679
IDispatch*(); 679
IsBound 679
Lookup 679
MustBeBound 680
SetLang 680
Unbind 680
Geschützter Konstruktor 680
Konstruktor 680
Geschützte Elementfunktion 680
Invoke 680
Struktur TAutoShort 681
Öffentliches Datenelement 681
ClassInfo 681
Klasse TAutoStack 681

Öffentlicher Konstruktor und Destruktor 682
Konstruktor 682
Destruktor 682
Öffentliche Elementfunktion 683
operator[] 683
Öffentliche Datenelemente 683
ArgCount 683
ArgSymbolCount 683
CurrentArg 683
LangId 683
Owner 683
Symbol 684
Konstante 684
SetValue 684
Struktur TAutoString 684
Öffentliche Konstruktoren und Destruktor 684
Konstruktoren 684
Destruktor 685
Öffentliche Elementfunktionen 685
int() 685
operator = ... 685
char*() 686
const char far*() 686
Öffentliches Datenelement 686
ClassInfo 686
Struktur TAutoType 686
Öffentliche Elementfunktion 687
GetType 687
Klasse TAutoVal 687
Öffentliche Elementfunktionen 688
operator = ... 688
TBool far*() 689
TBool() 689
Clear 689
Copy 689
double far*() 689
double() 689
float far*() 690
float() 690
GetDataType 690
IDispatch&() 690
IDispatch*() 690
int far*() 690
int() 690
IsRef 691
IUnknown&() 691
IUnknown*() 691
long far*() 691
long() 691
short far*() 691
short() 691
string() 692
far*() 692
TAutoCurrency() 692
TAutoCurrency far*() 692
TAutoDate far*() 692
TAutoDate() 692
TUString*() 692
unsigned long() 693
unsigned long far*() 693
Struktur TAutoVoid 693
Öffentliches Datenelement 693
ClassInfo 693
Typdefinition TComponentFactory 693
Typdefinition TLocaleId 694
Klasse TOcApp 695
Basisklasse 695
Typdefinitionen 695
TOcMenuEnable 695
Öffentliche Elementfunktionen 696
AddUserFormatName 696
Browse 696
Browse Clipboard 697
CanClose 697
Clip 697
Convert 698
Drag 698
EnableEditMenu 698
EvActivate 699
EvResize 699
EvSetFocus 699
GetName 699
GetNameList 700
GetRegistrar() 700
IsOptionSet 700
Paste 700
RegisterClass 701
RegisterClasses 701

ReleaseObject 701
SetOption 702
SetupWindow 702
TranslateAccel 702
UnregisterClass 703
UnregisterClasses 703
Geschützter Konstruktor und Destruktor 703
Konstruktor 703
Destruktor 704
Geschützte Elementfunktionen 704
ForwardEvent 704
Aufzählungstyp TOcAspect 704
Aufzählungstyp TOcAppMode 705
Aufzählungstyp TOcDialogHelp 706
Klasse TOcDocument 706
Öffentliche Konstruktoren und Destruktor 708
Konstruktoren 708
Destruktor 708
Öffentliche Elementfunktionen 708
Close 708
GetActiveView 708
GetName 709
GetParts 709
GetStorage 709
LoadParts 709
RenameParts 710
SaveParts 710
SaveToFile 710
SetActiveView 711
SetName 711
SetStorage 711
Struktur TOcDragDrop 712
Öffentliche Elementfunktionen 712
InitInfo 712
Pos 712
Where 712
Aufzählungstyp TOcDropAction 713
Klasse TOcFormatList 713
Öffentlicher Konstruktor und Destruktor 714
Konstruktor 714
Destruktor 714
Öffentliche Elementfunktionen 714
Add 714
Clear 714
Count 714
Detach 714
Find 715
IsEmpty 715
operator 715
Klasse TOcFormatListIter 715
Öffentlicher Konstruktor 716
Konstruktor 716
Öffentliche Elementfunktionen 716
Current 716
operator++ 716
operator int 716
Restart 716
Klasse TOcFormatName 717
Öffentliche Konstruktoren und Destruktor 717
Konstruktoren 717
Destruktor 717
Öffentliche Elementfunktionen 717
GetId 717
GetName 718
GetResult 718
operator = ... 718
Aufzählungstyp TOcInitHow 718
Klasse TOcInitInfo 718
Öffentliche Datenelemente 719
Container 719
HIcon 719
How 719
Storage 720
Where 720
Data 720
Path 720
Cld 720
Handle 721
Öffentliche Konstruktoren 721
TOcInitInfo 721
TOcInitInfo 721
Öffentliche Elementfunktionen 722
ReleaseDataObject 722
Aufzählungstyp TOcInitWhere 722
Aufzählungstyp TOcInvalidate 722
Struktur TOcMenuDescr 723

Öffentliche Datenelemente 724
HMenu 724
Width 724
Klasse TOcModule 724
Öffentlicher Konstruktor und Destruktor 724
Konstruktor 724
Destruktor 725
Öffentliche Elementfunktionen 725
GetRegistrar 725
IsOptionSet 725
OcInit 725
Öffentliche Datenelemente 726
OcApp 726
OleMalloc 726
Klasse TOcNameList 726
Öffentliche Konstruktoren und Destruktoren 727
Konstruktor 727
Destruktor 727
Öffentliche Elementfunktionen 727
operator [] 727
Add 727
Clear 728
Count 728
Detach 728
Find 728
IsEmpty 728
Klasse TOcPart 728
Basisklasse 728
Öffentliche Konstruktoren 729
Konstruktoren 729
Öffentliche Elementfunktionen 729
operator == 729
Activate 730
Close 730
Delete 730
Detach 730
DoVerb 730
Draw 731
EnumVerbs 731
GetName 731
GetNameLen 731
GetPos 732
GetRect 732
GetServerName 732
GetSize 732
IsActive 733
IsLink 733
IsSelected 733
IsVisible 733
Load 734
Open 734
Rename 734
Save 734
Select 735
SetActive 735
SetHost 735
SetPos 736
SetSize 736
SetVisible 736
Show 736
UpdateRect 737
Geschützter Destruktor 737
Destruktor 737
Klasse TOcPartCollection 737
Öffentlicher Konstruktor und Destruktor 737
Konstruktor 737
Destruktor 737
Öffentliche Elementfunktionen 738
Add 738
Clear 738
Count 738
Detach 738
Find 738
IsEmpty 738
Locate 739
SelectAll 739
Klasse TOcPartCollectionIter 739
Öffentlicher Konstruktor 740
Konstruktor 740
Öffentliche Elementfunktionen 740
operator ++ 740
Current 740
operator int 740
Restart 740
Aufzählungstyp TOcPartName 741
Klasse TOcRegistrar 741
Basisklasse 741

Öffentlicher Konstruktor und Destruktor 742
Konstruktor 742
Destruktor 742
Öffentliche Elementfunktionen 742
BOleComponentCreate 742
CreateOcApp 743
GetAppDescriptor 743
Geschützte Elementfunktionen 744
CanUnload 744
GetFactory 744
LoadBOle 744
Klasse TOcRemView 744
Basisklasse 744
Öffentlicher Konstruktor 745
Konstruktor 745
Öffentliche Elementfunktionen 746
Copy 746
EvClose 746
GetContainerTitle 746
GetInitialRect 746
Invalidate 746
IsOpenEditing 747
Load 747
Rename 747
Save 747
Struktur TOcSaveLoad 748
Öffentliche Elementfunktionen 748
Release 748
StorageI 748
Klasse TOcScaleFactor 748
Öffentliche Konstruktoren 749
Konstruktoren 749
Destruktor 750
Öffentliche Datenelemente 750
PartSize 750
SiteSize 750
Öffentliche Elementfunktionen 750
operator = ... 750
GetScale 751
GetScaleFactor 751
IsZoomed 751
SetScale 751
Aufzählungstyp TOcScrollDir 751
Struktur TOctool barInfo 752
Öffentliche Elementfunktionen 752
HBottomTB 752
HFrame 752
HLeftTB 753
HRightTB 753
HTopTB 753
Show 753
Klasse TOcVerb 753
Öffentlicher Konstruktor 754
Konstruktor 754
Öffentliche Datenelemente 754
CanDirty 754
TypeName 754
VerbIndex 755
VerbName 755
Klasse TOcView 755
Basisklasse 755
Öffentlicher Konstruktor 755
Konstruktor 755
Öffentliche Elementfunktionen 756
ActivatePart 756
BrowseClipboard 756
BrowseLinks 756
Copy 757
EvActivate 757
EvClose 757
EvResize 757
EvSetFocus 758
GetActivePart 758
GetOcDocument 758
GetOrigin 758
GetWindowRect 759
InvalidatePart 759
Paste 759
RegisterClipFormats 759
ReleaseObject 760
Rename 760
ScrollWindow 760
SetLink 760
SetupWindow 761
Geschützter Destruktor 761
Geschützter Destruktor 761
Geschützte Elementfunktionen 761
ForwardEvent 761
Init 761

Shutdown 762
Geschützte Datenelemente 762
ActivePart 762
Extent 762
FormatList 762
Link 762
OcApp 762
OcDocument 763
Origin 763
Win 763
WinTitle 763
Struktur TOcViewPaint 763
Öffentliche Datenelemente 764
Aspect 764
Clip 764
DC 764
Part 764
Pos 764
Klasse TOleAllocator 765
Öffentliche Konstruktoren und Destruktor 765
Konstruktor 765
Destruktor 765
Öffentliche Elementfunktionen 766
Alloc 766
Free 766
Öffentliches Datenelement 766
Mem 766
Klasse TRegistrar 766
Öffentlicher Konstruktor und Destruktor 767
Konstruktor 767
DestruktorDestruktor;TRegistrar 767
Öffentliche Elementfunktionen 768
CanUnload 768
CreateAutoApp 768
CreateAutoObject 768
GetFactory 769
GetOptions 769
IsOptionSet 769
ProcessCmdLine 770
ReleaseAutoApp 770
RegisterAppClass 770
Run 771
SetOption 771
Shutdown 771
UnregisterAppClass 772
Geschütztes Datenelement 772
AppDesc 772
Geschützter Konstruktor 772
Konstruktor 772
Klasse TUnknown 772
Öffentliche Elementfunktionen 773
Aggregate 773
GetOuter 773
GetRefCount 774
operator IUnknown&() 774
operator IUnknown*() 774
SetOuter 774
Geschützter Konstruktor und Destruktor 775
Konstruktor 775
Destruktor 775
Geschützte Elementfunktionen 775
QueryObject 775
ThisUnknown 775
Geschütztes Datenelement 775
Outer 775
Klasse TXAuto 776
Basisklasse 776
Öffentlicher Konstruktor 776
Konstruktor 776
Öffentliches Datenelement 776
ErrorCode 776
Typdefinition 776
TError 776
Klasse TXObjComp 777
Basisklasse 777
Öffentlicher Konstruktor 777
Konstruktor 777
Öffentliche Elementfunktion 777
ErrorCode 777
Typdefinition 777
TError 777
Klasse TXOle 778
Basisklasse 778
Öffentliche Konstruktoren und Destruktor 778
Konstruktoren 778
Destruktor 779

Öffentliche Elementfunktionen 779
Check 779
Öffentliches Datenelement 779
Stat 779
Klasse TXRegistry 780
Basisklasse 780
Öffentliche Konstruktoren 780
Konstruktoren 780
Öffentliche Elementfunktionen 780
Check 780
Key 781
Registrierschlüssel typehelp 781
Registrierschlüssel usage 781
Registrierschlüssel verb*n* 782
Registrierschlüssel verb*n*opt 782
Registrierschlüssel version 783
Botschaft WM_OCEVENT 783

Kapitel 7
Übersicht über ObjectSupport 785

Kapitel 8
ObjectSupport-Referenz 787

TLangId typedef 787
Registriermakros 787
Registriermakros 788
BEGIN_REGISTRATION-Makro 789
END_REGISTRATION-Makro 789
REGDATA-Macro 789
REGITEM-Makro 790
REGFORMAT-Makro 791
REGSTATUS-Makro 792
REGVERBOPT-Makro 792
REGICON-Makro 793
REGDOCFLAGS-Makro 794
REGISTRATION_FORMAT_BUFFER-Makro 794
Klasse TDocTemplate 795
Öffentliche Elementfunktionen 795
Geschützte Konstruktoren und Destruktoren 798
Klasse TDocTemplateT, <D,V> 799
Öffentliche Konstruktoren 800
Öffentliche Elementfunktionen 800
Klasse TDropInfo 801
Öffentliche Konstruktoren 801
Öffentliche Elementfunktionen 801
Klasse TPoint 802
Öffentliche Konstruktoren 803
Öffentliche Elementfunktionen 803
Klasse TPointer<> 805
Öffentliche Konstruktoren 806
Öffentliche Elementfunktionen 806
Klasse TProcInstance 806
Öffentliche Konstruktoren und Destruktor 807
Öffentliche Elementfunktionen 807
Klasse TRect 807
Öffentliche Konstruktoren 808
Öffentliche Elementfunktionen 809
Klasse TResId 814
Öffentliche Konstruktoren 815
Öffentliche Elementfunktionen 815
Friend-Funktionen 815
Klasse TSize 816
Öffentliche Konstruktoren 816
Öffentliche Elementfunktionen 817
Klasse TXBase 818
Öffentliche Konstruktoren und Destruktoren 819
Öffentliche Datenelemente 819
Öffentliche Elementfunktionen 819

Anhang A
Gekapselte WIN-API-Funktionen ... 821

Anhang B
Strukturen der Windows API 829

Struktur ABC 829
Struktur BITMAP 829
Struktur BITMAPCOREHEADER 831
Struktur BITMAPCOREINFO 831
Struktur BITMAPINFO 833
Struktur BITMAPINFOHEADER 834
typedef COLORREF 836
Struktur COMPAREITEMSTRUCT 836
Struktur DELETEITEMSTRUCT 837
Struktur DEVMODE 837

Struktur DRAWITEMSTRUCT 841
Struktur FINDREPLACE 842
Struktur GLYPHMETRICS 844
Struktur HANDLETABLE 844
Struktur ICONINFO 845
Struktur KERNINGPAIR 845
Struktur LOGBRUSH 846
Struktur LOGFONT 847
Struktur LOGPALETTE 850
Struktur LOGPEN 850
Struktur MDICREATESTRUCT 851
Struktur METARECORD 852
Struktur MEASUREITEMSTRUCT 852
Struktur MSG 853
Struktur OUTLINETEXTMETRIC 854
Struktur RGBQUAD 856
Struktur RGBTRIPLE 857
WNDCLASS struct 859
Struktur PAINTSTRUCT 861
Struktur PALETTEENTRY 862
Struktur XFORM 862

Anhang C
Vererbungsdiagramme 865

TApplication 865
TBitMap 866
TBitMapGadget 866
TButton 867
TButtonGadget 868
TCheckbox 869
TChooseColorDialog 870
TChooseFontDialog 872
TClipboardViewer 873
TComboBox 874
TComboBoxData 876
TCommonDialog 878
TControl 879
TControlBar 880
TControlGadget 881
TDecoratedFrame 882
TDecoratedMDIFrame 883
TDialog 885
TDocManager 886
TDocTemplate 886
TDocument 886
TEdit 887
TEditFile 889
TEditSearch 891
TEditView 892
TFileDocument 893
TFileOpenDialog 894
TFileSaveDialog 896
TFindDialog 898
TFloatingFrame 900
TGadgetWindow 902
TGauge 904
TLayoutWindow 905
TListBox 906
TListView 907
TMDIClient 908
TMDIFrame 909
TRadioButton 911
TSlider 913
TStatusBar 915
TTextGadget 916
TToolBox 917
TWindowView 919
TWindow 920

Index 921

Abbildungen

Abbildung 1.1:
Basisklasse mit mehreren abgeleiteten Klassen 33
Abbildung 1.2:
ObjectWindows Klassenhierarchie 34
Abbildung 1.3:
ObjectWindows Klassenhierarchie 35

Tabellen

Tabelle 1.2:
Übersicht über statische Bibliotheken und Importbibliotheken 43
Tabelle 1.3:
Übersicht über dynamische Linkbibliotheken 44
Tabelle 1.9:
Übersicht über Ressourcendateien 51

Tabelle 2.5:
Schalter-Flag-Konstanten 56
Tabelle 2.6:
Kommandobasierte Konstanten 57
Tabelle 2.7:
Kommandobasierte Konstanten 58
Tabelle 2.9:
Kommandobasierte Konstanten 58
Tabelle 2.10:
Kommandobotschafts- konstanten 59
Tabelle 2.11:
Dokumentverwaltungs-Modus-Konstanten 62
Tabelle 2.12:
Dokumentbotschaftsaufzählung 62
Tabelle 2.15:
ID-Dateikonstanten 66
Tabelle 2.16:
ID- Druckerkonstanten 66
Tabelle 2.17:
Tastenkürzel-ID-Konstanten 66
Tabelle 2.18:
Menü-ID-Konstanten 67
Tabelle 2.19:
Dokument-String-ID Konstanten 68
Tabelle 2.20:
Editierdatei-ID-Konstanten 68
Tabelle 2.21:
Exception-Botschaftskonstanten 68
Tabelle 2.22:
List-View String-ID-Konstanten 69
Tabelle 2.23:
Drucker-String-ID-Konstanten 70
Tabelle 2.24:
Prüfungs-ID-Konstanten 70
Tabelle 2.25:
shxxxx-Konstanten 82
Tabelle 2.26:
TCheckBox Markierungszustände 119
Tabelle 2.27:
Ereignisstatuskonstanten 263
Tabelle 2.28:
Zeichen Maskenformat 452
Tabelle 2.29:
Transferfunktions-konstanten 507
Tabelle 2.30:
Eigenschafts- und C++Typen 518
Tabelle 2.31:
Basic und C++ VBX-Datentypen 523
Tabelle 2.32:
VBX-Ereignisargumente 523
Tabelle 2.33:
Shifttastenbitwerte 524
Tabelle 2.34:
Maustastenargumente 524
Tabelle 2.36:
Vnxxxx View-Benachrichtigungs-IDs 581

Tabelle 3.1:
Schalter-Benachrichtigungsbotschaften . . . 583
Tabelle 3.2:
Benachrichtigungsbotschaften, die auf der ID untergeordneter Fenster basieren 583
Tabelle 3.3:
Kombinationsfenster-Benachrichtigungsbotschaften 584

Tabelle 3.4:
Kommandobotschaften 584
Tabelle 3.5:
Dokumentverwaltungsbotschaften 584
Tabelle 3.6:
Dokument-View-Botschaften 584
Tabelle 3.7:
Editierfeld-Benachrichtigungsbotschaften . 585
Tabelle 3.8:
Listenfeld-Benachrichtigungsbotschaften . . 585
Tabelle 3.9:
ObjectComponents-Botschaften 585
Tabelle 3.10:
Bildlaufleisten-Benachrichtigungs-
botschaften 586
Tabelle 3.11:
Windows-Standardbotschaften 587
Tabelle 3.12:
VBX-Dialogelementbotschaften 589
Tabelle 3.13:
Benutzerdefinierte Botschaften 590

Tabelle 7.1:
Auflistung der Bibliotheksdateien von
Object Support 785

Tabelle A.1:
Gekapselte *inline*-HWND-Funktionen 822
Tabelle A.2.:
Gekapselte Windows-Botschaften 822
Tabelle A.3:
Fensterkoordinaten und -größen 822
Tabelle A.4:
Fenstereigenschaften und -attribute 823
Tabelle A.5:
Fensterplazierung- und -anzeige 823
Tabelle A.6:
Fensterverhältnisse 824
Tabelle A.7:
Fensterzeichenfunktionen 824
Tabelle A.8:
Funktionen zum Blättern von Fenstern 825
Tabelle A.9:
Funktionen mit der ID untergeordneter
Fenster 825
Tabelle A.10:
Menü- und Menüleistenfunktionen 825
Tabelle A.11:
Zwischenablagefunktionen 826
Tabelle A.12:
Timer-Funktionen 826
Tabelle A.13:
Cursor- und Mauszeigerfunktionen 826
Tabelle A.14:
Tastenkürzelfunktionen 827
Tabelle A.15:
Hilfs- und Task-Funktionen 827

Einführung

Dieses *Referenzhandbuch* soll Ihnen bei den folgenden Aufgaben in ObjectWindows helfen:

- Nachschlagen von Informationen über den allgemeinen Zweck einer jeden Klasse.
- Erlernen der Details über die Verwendung einer besonderen ObjectWindows-Klasse und deren Elemente und Funktionen.
- Veranschaulichen der Beziehungen zwischen den ObjectWindows-Klassen bezüglich der virtuellen und der nicht-virtuellen Mehrfachvererbung.
- Lernen, welche Klassen Funktionen einführen oder redefinieren.
- Bestimmen, welcher Vorfahre einer Klasse ein Datenelement oder eine Elementfunktion eingeführt hat.
- Lernen, wie Datenelemente und Elementfunktionen deklariert werden. Verwenden der Ereignisbehandlungsfunktionen für das Antworten auf Botschaften.
- Verwenden der Zuteilungsfunktionen für das Extrahieren von Windows-Botschaften.

Inhalt dieses Handbuchs

Dieses Handbuch enthält sieben Referenzkapitel sowie drei Anhänge:

Kapitel 1: ObjectWindows im Überblick bietet eine Übersicht über ObjectWindows.

Kapitel 2: ObjectWindows-Referenz ist eine alphabetische Auflistung aller ObjectWindows-Standardklassen, einschließlich einer Erläuterung ihres Zwecks, ihrer Verwendung und ihrer Elemente. Die Referenz beschreibt zudem die Nicht-Objekt-Elemente, wie zum Beispiel Strukturen, Konstanten, Variablen und Makros, die von Klassen benutzt werden.

Kapitel 3: Ereignisbehandlungsroutinen listet die ObjectWindows-Funktionen und Benachrichtigungs-Codes auf, die Windows-Botschaften extrahieren.

Kapitel 4: Zuteilungsfunktionen listet alle ObjectWindows-Funktionen auf, die Windows-Botschaften zuteilen.

Kapitel 5: Übersicht über ObjectComponents bietet eine Übersicht über ObjectComponents.

Kapitel 6: Bibliotheksreferenz ObjectComponents ist eine Referenz zu ObjectComponents.

Kapitel 7: Übersicht über ObjectSupport bietet eine Übersicht zu ObjectSupport.

Kapitel 8: ObjectSupport-Referenz ist eine Referenz zu ObjectSupport.

Anhang A: Gekapselte WIN-API-Funktionen listet die ObjectWindows-Funktionen auf, die Windows-API-Funktionen kapseln.

Anhang B: Strukturen der Windows-API behandelt die Windows-API.

Anhang C: Vererbungsdiagramme listet die Elementfunktionen für jede Klasse auf und zeigt, welche Funktionen eine in der Basisklasse definierte Funktion überschreiben.

Symbole und Typographie in diesem Handbuch

Abhängig von der von Ihnen verwendeten Windows-Umgebung für die Anwendungsprogrammierung (API) sind auch unterschiedliche Object Windows-Funktionen verfügbar. Wenn ein Datenelement oder eine Funktion nur unter der Windows- 32-Bit- API einschließlich Windows- NT verfügbar ist, wird das Win32-Symbol links vom Datenelement oder der Funktion angezeigt.

Analog dazu wird das Win16-Symbol links vom Datenelement oder der Funktion angezeigt, wenn ein Klassenelement nur unter Win16 verfügbar ist. Alle Unterschiede in der Implementierung zwischen Win16 und Win32 werden beschrieben. Andernfalls gilt das Klassenelement als vollständig verfügbar, d.h. sowohl unter Win16- als auch unter Win32-APIs.

Fettschrift Fettschrift kennzeichnet Schlüsselwörter der Programmiersprache (wie z.B. **char, switch** und **begin**) sowie Kommandozeilenoptionen (wie z.B. **-rn**).

Kursiv Kursive Schrift wird bei Programmvariablen und Konstanten eingesetzt, die im Text erscheinen. Diese Typographie wird ebenso für die Hervorhebung bestimmter Wörter, wie beispielsweise neuer Begriffe, verwendet.

`Schreibmaschine` Schreibmaschine steht für Text, der in gleicher Form auf dem Bildschirm oder in einem Programm erscheint. Diese Schriftart wird auch für jede manuelle Eingabe (wie zum Beispiel `TD32` für das Starten des 32-Bit- Turbo Debugger) verwendet.

[*Taste*] Diese Typographie kennzeichnet eine Taste Ihrer Tastatur, wie beispielsweise in »Drücken Sie [*Esc*], um das Menü zu verlassen«.

[*Taste1*][*Taste2*] Tastenkombinationen, bei denen eine oder mehrere Tasten gleichzeitig gedrückt werden, werden durch [*Taste1*][*Taste2*] repräsentiert. Sie können zum Beispiel das Program-Reset-Kommando ausführen, indem Sie die Taste [*Strg*] gedrückt halten und die Taste [*F2*] drücken (repräsentiert durch [*Strg*][*F2*]).

Menü|Kommando Diese Kommandosequenz repräsentiert die Auswahl einer Menüoption. So repräsentiert zum Beispiel »*Datei / Öffnen*« das Kommando *Öffnen* im Dateimenü.

Dieses Symbol weist Sie auf besondere Informationen hin, die Sie beachten sollten.

Konventionen in diesem Handbuch

Inline-Funktionen, d.h. solche Funktionen, die innerhalb einer Klasse deklariert und definiert sind, weisen vor der Funktionsdeklaration das Schlüsselwort »inline« auf, z.B.:

```
inline virtual int AddString(const char far* string);
```

Cross-referenzierte Eintritte in ObjectWindows-Funktionen umfassen den Klassennamen, den Zugriffsoperator und den Funktionsnamen, z.B.:

TApplication::PumpWaitingMessages

Aufrufen von Windows-API-Funktionen geht der Zugriffsoperator (::) voran, z.B.:

::ShowWindow

C++ Datentypen, die Schlüsselwörter darstellen, werden klein und fett geschrieben (wie **int** und **long).** Vordefinierte Windows-Typen (wie HWND und UINT) stehen in Großbuchstaben, z.B.:

```
inline BOOL TrackPopupMenu(UINT flags;int x, int y, int rsvd,
                           HWND wnd, TRect* rect=0);
```

Object Windows Referenz

Teil I

K A P I T E L 1

ObjectWindows im Überblick

Dieses Kapitel bietet eine Übersicht der ObjectWindows Klassen, Bibliotheken und Header-Dateien. Es beschreibt die Klassen entsprechend den funktionellen Gruppen, wie sie im Hierarchiediagramm von ObjectWindows dargestellt sind.

Hierarchiediagramm

Das ObjectWindows Hierarchiediagramm zeigt die in diesem Handbuch beschriebenen Klassen. Sie sind nach funktionellen Kategorien in Gruppen eingeteilt und alle zusammengehörigen Klassen finden sich in einem schattierten Block des Diagramms. Ein Rechteck umschließt den Namen der Klasse. Im Falle einer Basisklasse einer Mehrfach-Vererbung ist der Name mit einer gestrichelten Linie eingerahmt. *TListBox* ist z.B. Basisklasse von *TListView*, die von *TView* ebenso wie von *TListBox* abgeleitet ist. Basisklassen sind oberhalb der abgeleiteten Klassen plaziert und mit diesen durch gerade Linien verbunden. Die auf den Verbindungslinien befindlichen Dreiecke kennzeichnen die Art der Vererbungsbeziehung zwischen den Klassen. Ein dunkles Dreieck zeigt virtuelle Vererbung zwischen Basis- und abgeleiteten Klassen an, ein helles Dreieck veranschaulicht nicht-virtuelle Vererbung.

Abbildung 1.1: Basisklasse mit mehreren abgeleiteten Klassen

Abbildung 1.2: ObjectWindows Klassenhierarchie

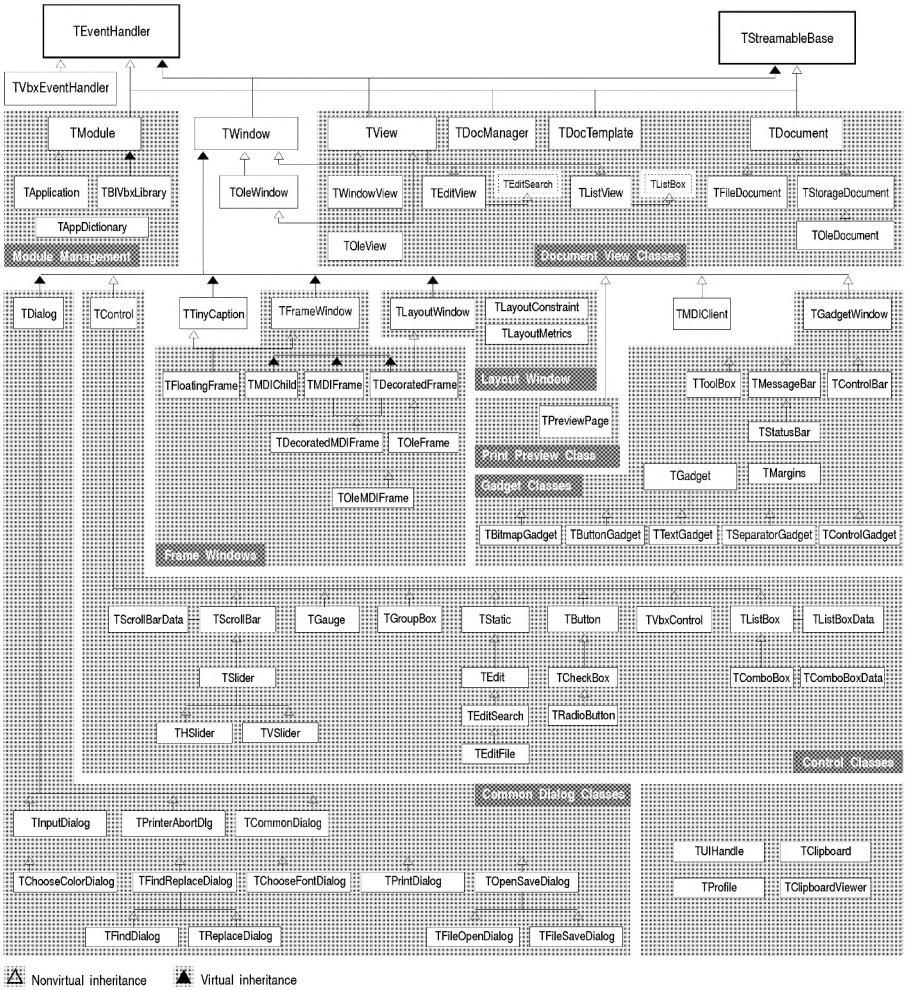
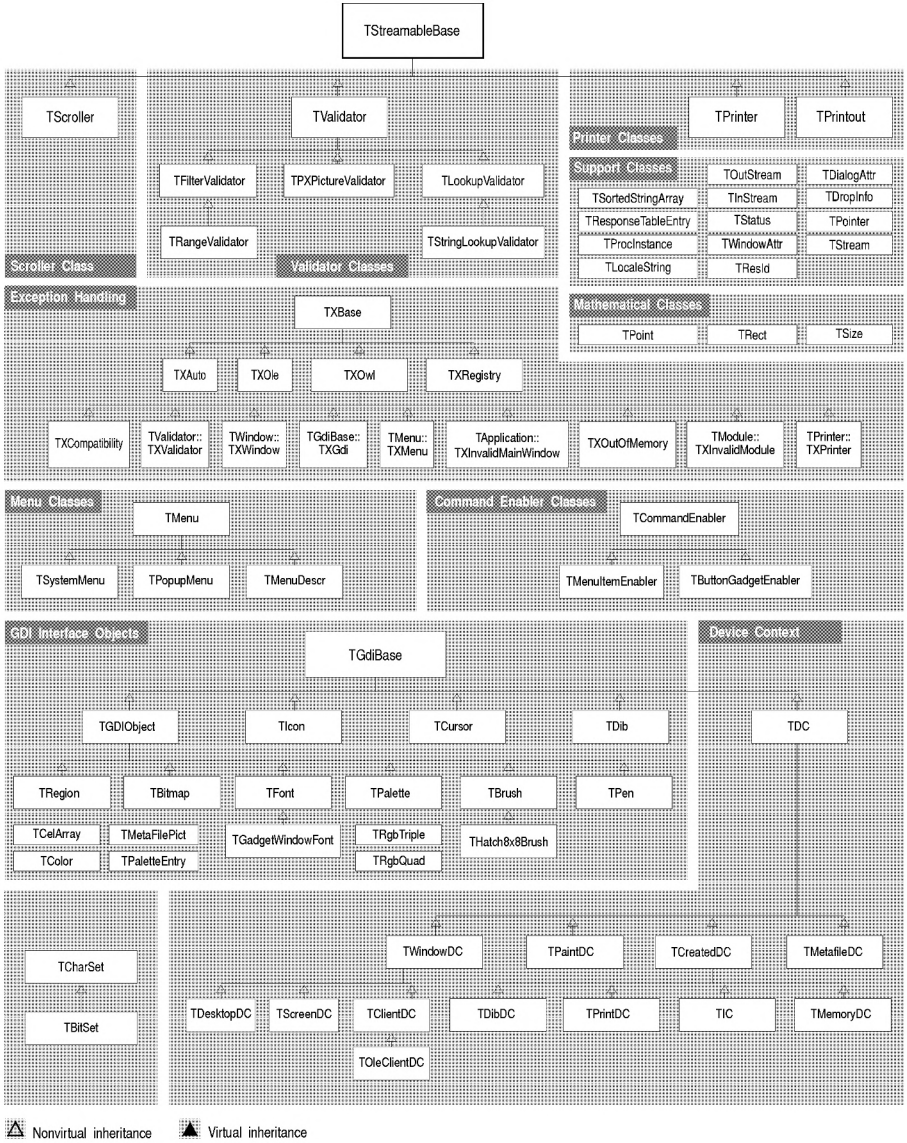


Abbildung 1.3: ObjectWindows Klassenhierarchie



Gebrauch der ObjectWindows Klassen

Die ObjectWindows Hierarchie umfaßt eine Unmenge von Klassen, die Sie verwenden, abändern, oder aus denen Sie ableiten können, um Ihre eigene Anwendung zu erstellen. Dieser Abschnitt beschreibt die Einteilung dieser Klassen und ihren Gebrauch beim Erstellen einer Anwendung. Die im Object Windows Hierarchiediagramm dargestellten Klassen lassen sich folgenden Gruppen zuordnen.

- Basisklassen
- Klassen zur Fensterverwaltung
  - Rahmenfenster
  - Dekorierte Fenster
  - Allgemeine Dialoge
  - Steuerelemente
  - Gadgets
  - Menüs
- Klassen zur Modul- und Programmverwaltung
  - Befehlsaktivierung
  - Dok/View
  - Drucken und Druckvoranzeige
- Klassen grafischer Objekte
  - GDI-Objekte
  - Gerätekontexte
- Prüfungsklassen
- Klassen zum Exception Handling
- Hilfsklassen

Basisklassen

TEventHandler, *TStreamableBase* und *TGdiBase* sind wichtige Basisklassen. Alle ObjectWindows Klassen sind von einer oder mehreren dieser Klassen abgeleitet. Von *TEventHandler* erbende Klassen können auf Windows-Botschaften antworten. Von *TStreamableBase* erbende Klassen unterstützen das Streaming, d.h., ihre Objekte können lesend und schreibend auf Streams zugreifen. Fast alle ObjectWindows Klassen lassen sich von *TStreamableBase* ableiten. Mehrfachvererbung kann zur gleichzeitigen Ableitung einer Klasse aus *TEventHandler* und *TStreamableBas*e genutzt werden. Von *TGdiBase*, einer privaten Basisklasse, erbende Klassen unterstützen GDI-Zeichenwerkzeuge wie etwa Stifte, Pinsel, Schriftarten und Bitmaps.

- *TEventHandler* schickt Botschaften an die zuständige Routine zur Botschaftsbehandlung

- *TStreamableBase* bietet Unterstützung für C++ Streaming und (Objekt-)Persistenz.
- *TGdiBase* stellt die Ursprungsklasse aller abgeleiteten GDI-Klassen dar, welche die Windows GDI-Bibliothek unterstützen.

Klassen zur Fensterverwaltung

Allen Fensterklassen übergeordnet ist *TWindow*, selbst abgeleitet von *TEventHandler* und *TStreamableBase*. Diese Klasse repräsentiert die allen Fenstern gemeinsame Funktionalität, seien es Dialogfelder, Steuerelemente, MDI- oder Layoutfenster. *TOleWindow*, eine der grundlegenden Object-Windows Klassen für die Implementierung der OLE-Funktionen, dient der Einbettung von Objekten in eine Verbunddokument-Anwendung.

Rahmenfenster

Ein Rahmenfenster, eigentlich das Hauptfenster einer Anwendung, kann andere Client-Fenster enthalten und bietet außerdem Unterstützung für Elemente der Benutzeroberfläche wie Menüs und Symbole. MDI-Rahmenfenster verwalten als Hauptfenster MDI-unterstützender Anwendungen mehrere Dokumente oder Fenster in einer Einzeldokument-Anwendung (siehe Single Document Interface SDI). ObjectWindows bietet auch OLE-Unterstützung für SDI- und MDI-Anwendungen. Ein bewegliches Rahmenfenster liefert dieselbe Funktionalität, läßt sich aber überall im übergeordneten Fenster anordnen.

- *TFrameWindow* erleichtert die Verwaltung von Hauptfenstern.
- *TFloatingFrame* ist abgeleitet aus *TFrameWindow* und *TTinyCaption* und bietet die Funtkionalität eines mit einer kleinen Titelleiste versehenen Rahmenfensters.
- *TMDIChild* definiert das Verhalten untergeordneter MDI-Fenster.
- *TMDIFrame* ermöglicht die Verwendung von Rahmenfenstern als MDI-Fenster.
- *TOleFrame* bietet OLE-Unterstützung für das Hauptfenster einer SDI-Anwendung.
- *TOleMDIFrame* bietet OLE-Unterstützung für das Hauptfenster einer MDI-Anwendung.

Gemischte Fensterklassen wie *TLayoutWindow* und *TClipboardViewe*r fügen den wichtigen Fensterklassen die Besonderheiten von Layout-Möglichkeiten und Zwischenablage hinzu. *TLayoutWindow* dient der Plazierung eines Fensters auf dem Bildschirm, während *TClipboardViewer* die Ansicht der von unterschiedlichen Anwendungen gemeinsam genutzten Daten ermöglicht.

Dekorierte Fenster

Durch Mehrfachvererbung aus *TFrameWindow* und *TLayoutWindow* abgeleitet, erlauben die Dekorfenster-Klassen die Ausschmückung von Dialogelementleisten und Statuszeilen bei Fensterrahmen und passen

dementsprechend das untergeordnete Fenster der Lage und Ausdehnung dieser Dekorationen an.

- *TDecoratedFrame* ist im wesentlichen ein Rahmenfenster mit schmückenden Zusätzen.
- *TDecoratedMDIFrame* repräsentiert ein MDI-Rahmenfenster mit Dekor.

Allgemeine Dialoge

TDialog erlaubt Ihnen das Anlegen spezieller Fenster, der sogenannten Dialogfelder. Der Benutzer wird von Dialogfeldern gewöhnlich nach Auskünften über Schriftarten, Farben, Dateien, Druckoptionen oder Angaben zur Textsuche und -ersatz gefragt.Je nach Zweck unterscheidet man zwischen modalen und nicht modalen Dialogfeldern. Im ersten Fall hindern sie den Benutzer an der Interaktion mit anderen Fenstern, im zweiten Fall erlauben sie es.

Sie können Ihre eigenen angepaßten Dialogfelder erzeugen oder eine der ObjectWindows Klassen verwenden, die die allgemeinen Windows Dialogfelder kapseln. Die folgenden Klassen für allgemeine Dialoge leiten sich ab von *TCommonDialog*, diese Klasse wiederum von *TDialog*, der Basisklasse für Dialogfelder.

- *TChooseFontDialog* repräsentiert modale Dialogfelder zur Selektion von Schriftart, -stil, -grad, -größe und -farbe.
- *TChooseColorDialog* repräsentiert modale Dialogfelder zur Farbenselektion und benutzerdefinierten Farbenanpassung.
- *TOpenSaveDialog* ist die Basisklasse modaler Dialogfelder zum Öffnen und Speichern einer Datei unter angegebenem Namen.
- *TPrintDialog* stellt ein modales Dialogfeld zum Drucken oder zur Druckereinstellung dar.
- *TFindReplaceDialog* ist die Basisklasse nicht-modaler Dialogfelder zur Textsuche und -ersatz.

Steuerelemente

Die Klassen zur Erzeugung von Steuerelementen unterstützen Windows Standardobjekte wie Listenfelder, Kombinationsfelder, Gruppenfelder, Markierungsfelder, Bildlaufleisten, Schaltfelder, Editierfelder und statische Elemente.

Obwohl die meisten Fensterobjekte mit Bildlaufleisten ausgestattet sind, kann man *TScrollBar* zum Anlegen einer vertikalen oder horizontalen Bildlaufleiste verwenden, etwa als ein Steuerelement eines Dialogfeldes.

Über den Windows Standard hinausgehend bietet ObjectWindows weitere sog. Hilfselemente an. Das sind spezialisierte, gänzlich in C++ implementierte Steuerelemente. Die ObjectWindows Klassen für Hilfselemente unterstützen Gleitregler, d.h. Steuerelemente zur Positionsanzeige ohne Bildlauf, und Meßanzeigen. Letztere sind Steuerelemente zur Darstel-

lung der Dauer eines oder sonstiger analoger Information über einen bestimmten Vorgang.

- *TSlider* definiert das grundlegende Verhalten der Gleitregler-Objekte
- *THSlider* implementiert horizontale Gleitregler.
- *TVSlider* implementiert vertikale Gleitregler.
- *TGauge* definiert die grundlegenden Eigenschaften der Meßanzeige-Objekte.

Gadgets

TGadget ist die Basisklasse mehrerer abgeleiteter Klassen von Gadget-Objekten mit Bezug zu einem Gadget-Fenster, eigenen Begrenzungen, Rändern und Eigenschaftsattributen. Die von *TWindow* abgeleitete Klasse *TGadgetWindow* verwaltete eine Menge von Gadgets, steuert deren Anzeige und schickt ihnen die nötigen Botschaften.

Zusätzliche von *TGadgetWindow* abgeleitete Klassen von Gadget-Objekten wie *TToolBox*, *TMessageBa*r, *TStatusBar* und *TControlBa*r manipulieren Gadgets auf verschiedene Weisen und erlauben so die Erweiterung einer Leiste oder Werkzeugpalette eines Rahmenfensters.

- *TToolBox* erlaubt die Matrixanordnung von Gadget-Objekten.
- *TMessageBar* implementiert eine Meldungszeile mit einem Text-Gadget.
- *TStatusBar* ermöglicht die Verwendung mehrerer Text-Gadgets mit unterschiedlichen Rahmenattributen innerhalb einer Textzeile.
- *TControlBar* realisiert einen Steuerbalken mit einer Leiste von Schaltern für ein Rahmenfenster.

Menüs

Die Klasse *TMenu* und ihre abgeleiteten Klassen gestatten die Konstruktion, Änderung und Erzeugung von Menü-Objekten. Zu den von *TMenu* abgeleiteten Klassen gehören

- *TPopupMenu* ermöglicht es, ein Popup-Menü einem existierenden Fenster oder Popup-Menü hinzuzufügen.
- *TSystemMenu* erzeugt ein Systemmenü-Objekt.

Klassen zur Modulverwaltung

Die von *TModule* abgeleitete Klasse *TApplication* stellt eine allen ObjectWindows Anwendungen gemeinsame Funktionalität zur Verfügung. Von *TApplication* abgeleitete Klassen haben die Eigenschaft, Instanzen einer Klasse und Hauptfenster erzeugen sowie Botschaften verarbeiten zu können. *TModule* definiert das Verhalten sowohl von Bibliotheks (DLL)- als auch Anwendungsmodulen. Die virtuell von *TModule* abgeleitete Klasse *TBiVbxLibrary* erlaubt die Verwendung von Visual Basic (VBX) Steuerelementen in einer Anwendung.

Klassen für das Dok/View-Modell

Das Dok/View-Modell meint ein System, in welchem die Verwaltung von und der Zugriff auf Daten mit Hilfe von Dokument-Objekten erfolgt,

während ein View-Objekt die Anzeige und Handhabung der Daten leistet. Eine beliebige Anzahl von View-Objekten kann mit einer bestimmten Dokumentart verknüpft sein. Verschiedene Klassen kontrollieren den Informationsfluß innnerhalb eines solchen Systems. Einige Klassen bieten außerdem Unterstützung für OLE Verbunddokumente und Verbunddateistrukturen innerhalb des Dok/View-Modells.

TDocManager ist die Basisklasse für den Umgang mit Dokumenten,Vorlagen, Botschaften usw.

- *TDocument* ist eine abstrakte Basisklasse, dient als Schnittstelle zwischen dem Dokumentobjekt, seinen View-Objekten und seinem Dokumentverwalter.
- *TStorageDocument* unterstützt die OLE Verbunddateistruktur und erlaubt das Erzeugen von Verbunddokumenten mit eingebundenen Objekten.
- *TOleDocument* implementiert den Dokumentanteil einer OLE-fähigen Dok/View-Anwendung.
- *TView* ist die Basisklasse zum Anzeigen von Dokumentdaten und Empfangen der Benutzereingaben.
- *TListView* unterstützt View-Objekte für Listenfelder.
- *TOleView* implementiert den View-Anteil einer OLE-fähigen Dok/View-Anwendung.

Drucker und Drucken

TPrinter, TPrintout und *TPreviewPage* liefern verschiedene Funktionen zur Vereinfachung von Vorgängen wie der Druckereinrichtung mit Hilfe eines Dialogfeldes, der Seitenansicht eines Dokumentes in einem Druckvoranzeige-Fenster, oder dem Ausdrucken eines Dokumentes.

- *TPrinter* repräsentiert das physische Druckerobjekt.
- *TPrintout* stellt das physisch auszudruckende Dokument dar, wie es zum Drucker übermittelt wird.
- *TPreviewPage* zeigt eine Seite eines Dokumentes in einem Druckvoranzeige-Fenster.

Klassen von Grafikobjekten

ObjectWindows Klassen von GDI-Objekten verkapseln die Windows Graphische Geräteschnittstelle (GDI) und vereinfachen damit den Gebrauch der Gerätekontext- und GDI-Objekte. Die GDI-Bibliothek unterstützt geräteunabhängiges Zeichnen unter Verwendung von DIBs (geräteunabhängige Bitmaps).

DC-Klassen

Anstatt grafische Information direkt einem Gerät (wie einem Bildschirm oder einem Drucker) zu übergeben, kann man mit Hilfe von GDI-Klassen diese Information in einer Bitmap-Darstellung unter Verwendung eines Gerätekontextes (DC) speichern. Ein *Gerätekontex*t ist eine Struktur, wel-

che Informationen über zeichnerische Attribute eines bestimmten Gerätes (wie etwa Stift, Pinsel, Text, Farben etc.) bereithält. DC-Klassen erlauben eine Vielfalt von Operationen seitens der Gerätekontext-Objekte.

- *TDC* ist die Stammklasse der Klassen zur Verpackung der GDI DC-Objekte.
- *TWindowDC* und abgeleitete Klassen wie *TClientDC* und *TScreenDC* erlauben den Zugriff auf den zu einem Fenster gehörigen Bereich.
- *TCreatedDC* und abgeleitete Klassen erlauben den Zugriff auf verschiedene Gerätekontexte, die ähnlich erzeugt und gelöscht werden wie Speicherplatz- und Druckobjekte.

GDI-Klassen

Die ObjectWindows Grafikbibliothek enthält mehrere Klassen zur Erzeugung von Objekten wie DIBs, Pinsel, Paletten, Stifte und andere Zeichenwerkzeuge.

- *TGdiBase* ist die private Basisklasse, von der die Klassen *TGDIObject*, *TIcon*, *TCurso*r und *TDib* abgeleitet sind.
- *TGDIObject* ist eine Basisklasse von mehreren anderen GDI-Klassen zur Unterstützung von Zeichenwerkzeugen.
- *TDib* verkapselt Strukturen zur Verwaltung von Formatierungs- und Paletteninformationen.
- *TCursor* verkapselt GDI-Zeigerobjekte

Prüfungsklassen

TValidator bildet die Basisklasse von verschiedenen ObjectWindows Klassen zur Verkapselung von Prüfungsobjekten. Die folgenden abgeleiteten Klassen erleichtern es Ihnen, Datenprüfungen in Ihre Anwendungen einzufügen.

- *TFilterValidator* und die daraus abgeleitete Klasse *TRangeValidator* prüfen Benutzereingaben für ein Eingabefeld während der Eingabe auf Gültigkeit.
- *TPXPictureValidator* vergleicht Benutzereingaben mit einer Datenformatmaske.
- *TLookupValidator* gleicht einen vom Benutzer eingegebenen String mit einer Liste gültiger Werte ab.

Klassen zum Exception Handling

Klassen zum Exception Handling stellen verschiedene Funktionen bereit, die Ihnen bei der Anfertigung fehlerfreier ObjectWindows Anwendungen behilflich sind. TXBase ist die Basisklasse aller ObjectWindows Klassen und ObjectComponent Klassen. Von *TXBase* abgeleitet, ist die Klasse *TXOw*l die Basisklasse der folgenden ObjectWinodws Klassen zum Exception Handling:

- *TXCompatibility* existiert aus Gründen der Rückwärtskompatibilität mit Code der ObjectWindows Version 1.0.

- *TXOutOfMemory* beschreibt Exceptions, die sich als Folge nicht genügend zur Verfügung stehenden Arbeitsspeichers ergeben.
- Geschachtelte Klassen zum Exception Handling wie *TXInvalidmainWindow*, *TXInvalidModule*, *TXWindow*, *TXMenu*, *TXValidator*, *TXGd*i und *TXPrinte*r beziehen sich auf bestimmte Fehlerbedingungen, die sich etwa ergeben, wenn ein Hauptfenster, ein Modul, ein Menüobjekt, ein Prüfungsobjekt, ein GDI-Objekt oder ein Drucker-Gerätekontext ungültig sind.

Befehlsaktivierungsklassen

Obwohl einige ObjectWindows-Klassen Befehle ausführen, gibt es allerdings drei Klassen, die speziell der Aktivierung bzw. Deaktivierung der einer Anwendung verfügbaren Befehlsoptionen dienen.

- *TCommandEnabler* ist die Basisklasse der folgenden beiden Klassen.
- *TButtonGadgetEnabler* aktiviert und deaktiviert Schalter-Gadgets.
- *TMenuItemEnabler* aktiviert und deaktiviert Menüoptionen und plaziert dementsprechend Checkmarks neben den Menüeinträgen.

ObjectSupport-Bibliotheksklassen

Klassen dieser Kategorie stellen verschiedene Dienste bereit, die Ihnen beim Entwurf Ihrer ObjectWindows Anwendung behilflich sein können. Die Klasse *TLocaleString* z.B. lokalisiert OLE-Registrierungsinformation, die für Container und Server erforderlich ist. Diese Klassen umfassen folgende Gruppen:

- Klassen mathematischer Objekte (wie *TPoint, TSize, TRect*) zur Beschreibung von Bildschirmkoordinaten und der Eigenschaften von Rechtecken.
- Klassen zur Registrierung und Lokalisierung (wie *TRegList* und *TLocaleString*) zur Vereinfachung der Registrierung von OLE-Containern und -Servern.
- Klassen von Dokumentvorlagen zur Vereinfachung des Entwurfs von Dok/View-Anwendungen

Die folgende Tabelle listet die Dateien der Hilfsbibliothek (..\OSL Verzeichnis).

Tabelle 1.1: Zusammenfassung der Dateien der ObjectSupport Hilfsbibliothek

Dateiname	Klassendefinition	Verwendung
defs.h		Enthält allgemeine Definitionen. einschließlich windows.h, und befaßt sich mit BOOLschen Datetypen
doctpl.h	TDocTemplate TDocTemplateT<D,V>	Erzeugt die Dok/View-Klassen Registriert die assoziierten Dokument- und View-Klassen
except.h	TXBase	Basisklasse des Exception Handlings für ObjectWindows bzw ObjectComponent Klassen
geometry.h	TDropInfo	Unterstützt Ziehen und Ablegen für Dateinamen

	Tpoint, TSize, TRect	Klassen mathematischer Objekte
	TPointer	Stellt Exception-freie Zeigerhandhabung bereit
	TProcInstance	Ein Klasse für Win16-Anwendungen
	TResId	eine Resourcen-ID
locale.h	TLocaleString	Lokalisierbarer Ersatz für char*
	TRegItem	Ein Eintrag des Systemregisters
	TRegList	Liste von Registereinträgen

Die ObjectWindows-Bibliotheken

Die folgenden Tabellen listen die statischen und dynamischen Bibliotheken von ObjectWindows, ihre Verwendung sowie das Betriebssystem, unter dem die Bibliotheken verfügbar sind, auf. Diese Dateien finden sich in Ihrem Bibliotheksverzeichnis.

Der Name der Datei OWLWx.LIB hängt von verschiedenen Faktoren ab – ob Sie eine Anwendung mit Speichermodell *Small*, *Medium* oder *Large* oder ob Sie eine WIN16- oder WIN32-Anwendung erzeugen wollen. Wenn die Anwendung zum Beispiel für 16-Bit mit Speichermodell *Small* erzeugt werden soll, ist der Name der Bibliotheksdatei OWLWS.LIB. Wenn Sie eine *Flat*-Model WIN32-Anwendung erzeugen wollen, ist der Name der Bibliotheksdatei OWLWF.LIB. Dabei zeigt »F« an, daß es sich um eine Anwendung im Flat-Modell handelt.

Grundlegende Versionen dieser ObjectWindows Dateien sind auf Ihrer Installationsdiskette enthalten. Sie können diese zusätzlichen Versionen erzeugen, indem Sie die ObjectWindows Make-Datei zu Hilfe nehmen, die sich in Ihrem Unterverzeichnis ..\EXAMPES befindet, und dabei die Schalter -DDIAGS und -DMODEL benutzen.

Tabelle 1.2: Übersicht über statische Bibliotheken und Importbibliotheken

Dateiname	Anwendung	Verwendung
OWLWS.LIB	Win16	16-Bit Speichermodell Small
OWLWM.LIB	Win16	16-Bit Speichermodell Medium
OWLWL.LIB	Win16	16-Bit Speichermodell Large
OWLDWS.LIB	Win16	16-Bit Diagnoseversion Speichermodell Small
OWLDWM.LIB	Win16	16-Bit Diagnoseversion Speichermodell Medium
OWLDWL.LIB	Win16	16-Bit Diagnoseversion Speichermodell Large
OWLWF.LIB	Win32s, Win32	32-Bit Bibliothek
OWLDWF.LIB	Win32, Win32s	32-Bit Diagnosebibliothek
OWLWI.LIB	Win16	16-Bit Importbibliothek für OWL200.DLL
OWLDWI.LIB	Win16	16-Bit Importbibliothek für OWL200D.DLL

OWLWFI.LIB	Win32, Win32s	32-Bit Importbibliothek für OWL200F.DLL
OWLDWFI.LIB	Win32, Win32s	32-Bit Importbibliothek für OWL200DF.DLL

Die DLL-Versionen von ObjectWindows sind im Unterverzeichnis BIN der Installation enthalten. Die folgende Tabelle listet die DLL-Namen, die Verwendungen sowie das Betriebssystem, unter dem jede Bibliothek verfügbar ist, auf.

Tabelle 1.3: Übersicht über dynamische Linkbibliotheken

Dateiname	Anwendung	Verwendung
OWL200.DLL	Win 16	16-Bit dynamische Bibliothek
OWL200F.DLL	Win 32	32-Bit dynamische Bibliothek
OWL200D.DLL	Win 16	Diagnoseversion der 16-Bit dynamischen Bibliothek
OWL200DF.DLL	Win 32	Diagnoseversion der 32-Bit dynamischen Bibliothek

Compiler-Optionen zur Erstellung und Benutzung der ObjectWindows Bibliotheken.

Sie benötigen unterschiedliche Compiler-Optionen, je nachdem, ob Sie DLL- bzw. statische ObjectWindows Bibliotheken erstellen oder benutzen wollen. Soweit Sie es nicht anders angeben, macht ObjectWindows gewisse Annahmen über Standardeinstellungen für Systemumgebungen und Speichermodelle. Als Standard-Systemumgebung nimmt ObjectWindows zunächst Win16, sofern nicht der MODEL-Schalter explizit auf “f” gesetzt ist und ObjectWindows dann Win32 als Umgebung festlegt. Die Standardeinstellung des MODEL-Schalters ist “d” und zeigt damit an, daß Sie die DLL-Version einer Bibliothek erstellen wollen.

Die folgende Tabelle listet die möglichen Kombinationen von Einstellungen der Schalter SYSTEM- und MODEL auf, die Sie für die Erstellung einer bestimmten Zielanwendung einsetzen können.

Tabelle 1.4: Zielanwendungen und Compiler-Optionen

Für diese Zielanwendung:	Setze SYSTEM =	Setze MODEL =
16-bit Windows-Anwendung , statisch, im Small-Modell	WIN16	s
16-bit Windows-Anwendung, statisch, im Medium-Modell	WIN16	m
16-bit Windows -Anwendung, statisch, im Compact-Modell	WIN16	c
16-bit Windows-Anwendung, statisch, im Large-Modell	WIN16	l
16-bit Windows-Anwendung DLL, Large-Modell	WIN16	d
32-bit Windows-Anwendung, statisch	WIN32	f
32-bit Windows-Anwendung, DLL	WIN32	d

Erstellen der ObjectWindows Bibliotheken

Falls Sie ObjectWindows DLLs und andere Bibliotheken erstellen, brauchen Sie mehrere vordefinierten Macros. Zum Beispiel dient das MAKE-

Macro USERDLL dazu, ObjectWindows in einer Benutzer-DLL zu verwenden, und hängt dabei den Suffix "U" an den Bibliotheksnamen. Das für die Erstellung einer ObjectWindows DLL zu definierende Präprozessor-Macro _BUILDOWLDLL legt Werte fest für die Macros _OWLCLASS, _OWLDATA und _OWLFUNC.

Die folgende Tabelle listet die MAKE-Optionen zur Erstellung von 16-bit oder 32-bit ObjectWindows Anwendungen. Sie können das Speichermodell entweder mit s (Small), m (Medium), l (Large), f (Flat), oder d (DLL) festlegen. Die von Ihnen gesetzten MAKE-Optionen sind dann verantwortlich für die Erzeugung des entsprechenden Präprozessor-Macros, welches seinerseits die angegebenen Werte für die _OWLCLASS-, _OWLDARA-, _OWLFUNC-Macros erzeugt und die geeignete Bibliothek erstellt. Ein Beispiel für die Verwendung dieser Einstellungen finden Sie im Makefile im Verzeichnis SOUCE\OWL oder in der Header-Dateo owldefs.h im Verzeichnis INCLUDE\OWL.

Tabelle 1.5: Zusammenfassung der Möglichkeiten zur Erstellung einer statischen oder dynamischen ObjectWindows Bibliothek

Zur Erstellung einer:	Verwenden Sie diese MAKE-Optionen	Präprozessor-Macro:	_OWLCLASS,_OWLDATA ,_OWLFUNC	Bibliotheken
ObjectWindows-DLL				
16-bit EXE	MODEL = d	_BUILDOWLDLL	alle als __export definiert	OWLWI.LIB
32-bit EXE oder DLL	MODEL = d -DWIN32	_BUILDOWLDLL	alle als __export definiert	OWLWFI.LIB
16-bit DLL	MODEL = d -D USERDLL	_BUILDOWLDLL	alle als __export definiert	OWLWIU.LI
Statische ObjectWindows Bibliothek				
16-bit EXE	MODEL = s | m | l	Keines	–	OWLWS.LIB, OWLWM.LIB, OWLWL.LIB
32-bit EXE oder DLL	MODEL = f -DWIN32	Keines	–	OWLWF.LIB
16-bit DLL	MODEL = l -DUSERDLL	Keines	–	OWLWLU.DLL

Verwendung der ObjectWindows Bibliothken

Diese Tabelle führt diejenigen MAKE-Optionen auf, die Sie zur Verwendung von 16-bit oder 32-bit ObjectWindows Anwendungen benötigen. Sie können für das Speichermodell zwischen den Einstellungen s (Small), m (Medium), l (Large), f (Flat) oder d (DLL) wählen. (Denken Sie daran, daß die MAKE-Optionen SYSTEM = WIN32 und -DWIN32 dasselbe bedeuten.) Die von Ihnen gesetzten MAKE-Optionen sind dann verantwortlich für die Erzeugung des entsprechenden Präprozessor-Macros, welches seinerseits die angegebenen Werte für die _OWLCLASS-, _OWLDARA-, _OWLFUNC-Macros erzeugt und die geeignete Bibliothek erstellt. Ein

Beispiel für die Verwendung dieser Einstellungen finden Sie in der Datei MAKEFILE.GEN im Verzeichnis \OWL\EXAMPLES.

Tabelle 1.6: Zusammenfassung der Optionen für den Einsatz einer statischen oder dynamischen ObjectWindows Biböliothek glichkeitenk

Zur Erstellung einer:	Verwenden Sie diese MAKE-Optionen	Präprozessor-Macro:	_OWLCLASS,_OWLDATA ,_OWLFUNC	Bibliotheken
ObjectWindows-DLL				
16-bit EXE	MODEL = d	_OWLDLL	alle als __import definiert	OWLWI.LIB
32-bit EXE oder DLL	MODEL = d -DWIN32	_OWLDLL	alle als __import definiert	OWLWFI.LIB
16-bit DLL	MODEL = d	_OWLDLL	alle als __import definiert	OWLWIU.LI
Statische ObjectWindows Bibliothek				
16-bit EXE	MODEL = s | m | l	Keines	–	OWLWS.LIB, OWLWM.LIB, OWLWL.LIB
32-bit EXE oder DLL	MODEL = f -DWIN32	Keines	–	OWLWF.LIB
16-bit DLL	MODEL = l Keines	–	OWLWLU.DLL	

Die folgende Tabelle listet die MAKE- bzw. Compiler-Optionen für das OWLFARVTABLE-Macro, welches die ObjectWindows Tabellen virtueller Funtkionen aus dem DGROUP-Teil des Datensegmentes heraus in das Code-Segment verschiebt.

Tabelle 1.7: Compiler-Optionen für das _OWLCLASS-Macro

Bei Verwendung dieser Optionen:	Und dieser Definition in Ihrem Makefile:	Erhalten Sie:
_OWLFARVTABLE, _BIDSFARVTABLE, _RTLFARVTABLE	OWLFARVABLE	den Zusatz __huge zum _OWLCLASS -Macro im statischen Fall
_FASTTHIS	n.a.	den Zusatz _fastthis zum _OWLCLASS-Macro

Die ObjectWindows-Header-Dateien

Header-Dateien enthalten Prototypdeklarationen für Klassenfunktionen sowie Definitionen für Datentypen und symbolische Konstanten.

Tabelle 1.8: Übersicht über Header-Dateien

Dateiname	Klassendefinition	Verwendung
Verzeichnis OWL\INCLUDE		
appdict.h	TAppDictionary	Enthält eine Menge aus Verbindungen zwischen einer Anwendung und einer Prozeßkennung.
applicat.h	TApplication	Kontrolliert das grundlegende Verhalten aller ObjectWindows-Anwendungen.
bitmapga.h	TBitMapGadget	Eine Menge, die nicht mehr wie 256 Einträge-mittels Bits verwaltet.
bitset.h	TBitSet TCharSet	Setzt oder löscht eines oder mehrere Bits. Eine Menge von Zeichen.
button.h	TButton	Erzeugt verschiedene Typen von Schalter-Dialogelementen.
buttonga.h	TButtonGadget	Erzeugt Schalter-Gadgets, die an- oder ausgeklickt werden können.
celarray.h	TCelArray	Erzeugt ein Array mit Cels.
checkbox.h	TCheckBox	Repräsentiert ein Markierungsfeld-Dialogelement.
chooseco.h	TChooseColor	Repräsentiert modale Dialogfenster, die eine Farbauswahl ermöglichen.
choosefo.h	TChooseFont	Repräsentiert modale Dialogfenster, die eine Schriftauswahl ermöglichen.
clipboar.h	TClipboard	Enthält Funktionen, die die Behandlung von Daten in der Zwischenablage kontrollieren.
clipview.h	TClipboardViewer	Registriert ein TClipboardViewer als einen Betrachter der Zwischenablage.
color.h	TColor	Enthält Funktionen für einfache Windows-Standard-farboperationen.
combobox.h	TCombobox	Erzeugt Kombinationsfenster in einem Fenster sowie die Klasse TComboBoxData für den Transfer von Daten zwischen Kombinationsfenstern.
commdial.h	TCommonDialog	Abstrakte Basisklasse für TCommonDialog-Objekte.
compat.h		Definiert Funktionen und Konstanten, die von ObjectWindows intern verwendet werden.
control.h	TControl	Verwendet für die Erzeugung von Dialogelement-objekten in abgeleiteten Klassen.
controlb.h	TControlBar	Implementiert eine Dialogelementleiste, die mnemonischen Zugriff für ihre Schalter-Gadgets bereitstellt.
controlg.h	TControlGadget	Ermöglicht das Plazieren von Dialogelementen in einem Gadgetfenster.
dc.h	TBandInfo, TClientDC, TCreatedDC, TDC, TDesktopDC, TDibDC TIC, TMemoryDC, TMetaFileDC, TPaintDC TPrintDC, TScreenDC,	GDI DC-Rahmenklasse, die DC-Objekte erzeugt.

	TWindowDC	
decframe.h	TDecoratedFrame	Erzeugt ein Client-Fenster, in das Dekorationen plaziert werden können.
decmdifr.h	TDecoratedMDIFrame	Erzeugt ein Rahmenobjekt, das dekorierte untergeordnete Fenster unterstützt.
dialog.h	TDialog	Erzeugt modale und nicht-modale Dialogfenster-Oberflächenelemente.
	TDialogAttr	Enthält die Attribute der Dialogfensterelemente.
dispatch.h		führt Funktionen aus, die Windows-Botschaften aufspalten von Windows-Botschaften.
docmanag.h	TDocManager	Erzeugt ein Dokumentverwaltungsobjekt, das Dokumente und Templates verwaltet.
	TDocTemplate	Erzeugt die Templates.
doctpl.h	TDocTemplate	Erzeugt die Templates.
	TDocTemplateT	Registriert die assoziierten Dokument- und View-Klassen.
docview.h	TDocument, TView, TWindowView, TStream, TInStream, TOutStream	Erzeugen, Entfernen und Senden von Botschaften im Zusammenhang mit Dokument-Views.
@TABN OL = edit.h	TEdit	Erzeugt ein Editierfeld-Oberflächenelement.
editfile.h	TEditFile	Erzeugt ein Datei-Editierfenster.
editsear.h	TEditSearch	Erzeugt ein Editierfeld, das auf Suchen- und Ersetzen-Kommandos reagiert.
editview.h	TEditView	View-Wrapper für TEdit.
eventhan.h	TEventHandler	Verwendet zur Ableitung von Klassen, die zur Behandlung von Botschaften fähig sind.
except.h	TXOwl	Basisklasse des Exception-Handlings.
	TXBase	
	TXCompatibility,	Dient der Abwärtskompatibilität.
	TXOutOfMemory,	Beschreibt eine Exception des Typs unzureichender Arbeitsspeicher.
	TStatus	Beschreibt eine Status-Exception.
filedoc.h	TFileDocument	Öffnet und schließt Dokument-Views.
findrepl.h	TFindDialog, TFindReplaceDialog::	Diese Klassen erzeugen und definieren die Attribute von nicht-modalen Dialogfenstern, die auf Suchen- und Ersetzen-Kommandos reagieren.
floatfra.h	TFloatingFrame	Implementiert einen fließenden Rahmen innerhalb eines übergeordneten Fensters.
framewin.h	TFrameWindow	Kontrolliert fensterspezifisches Verhalten, wie zum Beispiel Tastatur-bearbeitung und Kommandoverarbeitung.
gadget.h	TGadget	Erzeugt Gadgetobjekte, die zu einem Gadgetfenster gehören und spezifizierte Attribute besitzen.
gadgetwi.h	TGadgetWindow	Verwaltet eine Liste mit nebeneinander angeordneten Gadgets für ein Fenster.
	TGadgetWindowFont	Definiert die in Gadget-Fenstern verwendete Schriftart.
	TSeparatorGadget	Erzeugt eine Trennlinie zwischen Gadgets.
gauge.h	TGauge	Etabliert das Verhalten von Meßanzeigen-Dialogelementen.
gdibase.h	TGdiBase	Abstrakte Basisklasse für alle GDI-Klassen.
gdiobjec.h	TGdiObject	GDI-Basisklasse.

	TPen, TBrush, TFont, TPalette, TBitmap	Diese Klassen erzeugen spezifizierte GDI-Objekte: TIcon, TCursor, TDib und TRegion.
geometry.h	TPoint, TSize, TRect	Mathematische Klassen.
	TDropInfo	Unterstützt Dateinamenoperationen über das Ziehen der Maus
	TProcInstance	Hilfsklasse für Win16.
	TPointer	Stellt Exception-sichere Zeigerbehandlung zur Verfügung.
	TResId	Erzeugt eine Ressourecn-ID.
groupbox.h	TGroupBox	Erzeugt ein Gruppenfeldobjekt, das ein Gruppen-feldelement in Windows repräsentiert.
inputdia.h	TInputDialog	Generisches Dialogfenster.
keymodet.h	TKeyboardModeTracker	Eine Mischklasse für die Überwachung von Änderungen bei Tastaturmodi.
layoutco.h	TLayoutConstraint	Erzeugt Layout-Restriktionen.
layoutwi.h	TLayoutMetrics	Enthält die Layout-Restriktionen für die Definition der Layout-Maße eines Fensters.
listbox.h	TListBox	Erzeugt ein Listenfeldobjekt.
	TListBoxData	Verwendet für den Transfer des Inhalts eines Listenfeldes.
listview.h	TListView	Stellt Views für Listenfelder bereit.
locale.h	TLocaleString	Lokalisierbarer Ersatz für char*.
mdi.h	TMDIClient	Verwaltet untergeordnete MDI-Fenster.
	TMDIFrame	Das Hauptfenster bei MDI-verträglichen Anwendungen.
mdichild.h	TMDIChild	Definiert das Verhalten untergeordneter MDI-Fenster.
menu.h	TMenu, TPopupMenu,	Erzeugen von Menüobjekten.
	TSystemMenu	Basismenüklasse.
	TMenuDescr	Menüleiste mit Gruppen.
messageb.h	TMessageBar	Implementiert eine Meldungszeile.
metafile.h	TMetaFilePict	Eine Rahmenklasse, die mit TMetaFileDC verwendet wird.
module.h	TModule	Definiert das grundlegende Verhalten für ObjectWindows-Bibliotheken und -Anwendungen.
oledoc.h	TOleDocument	Implemetiert den Dokumentteil des Dok/View-Paares.
olefacto.h	TAutoFactory<<>>	Klassen-Template, das Callback-Funktionen für die Komponentenerzeugung in automatisierten OLE-fähigen Anwendungen unterstützt.
oleframe.h	TOleFrame	Unterstützt OLE-fähige Hauptfenster für SDI-Anwendungen.
olemdifr.h	TOleMDIFrame	Unterstützt OLE-fähige Hauptfenster für MDI-Anwendungen.
oleview.h	TOleView	Unterstützt den View-Teil des Dok/View-Paares.
olewindo.h	TOleClientDC	Übersetzt zwischen zwei verschiedenen Koordinatensystemen.
opensave.h	TOpenSave	Basisklasse für modale Dialogfenster für Öffnen und Sichern.
owlall.h		Include-Datei für alle ObjectWindows-Klassen.
owlcore.h		Include-Datei für die ObjectWindows-Kernklassen.
owldefs.h		Enthält Definitionen der Makros, die von allen ObjectWindows-Programmen verwendet werden.
owlpch.h		Enthält Definitionen von Makros, Daten und Funktionen für ObjectWindows.

preview.h	TPreviewPage	Zeigt eine Dokumentseite in einem Druckvorschau-fenster an.
printdia.h	TPrintDialog	Zeigt ein modales Dialogfenster zum Drucken oder zur Druckereinrichtung an.
printer.h	TPrinter TPrintout TPrinterAbortDlg	Repräsentiert das Druckergerät. Repräsentiert das gedruckte Dokument. Repräsentiert das Dialogfenster zum Druckabbruch.
radiobut.h	TRadioButton	Erzeugt ein Schaltfeld-Dialogelement.
scrollba.h	TScrollBar TScrollBarData	Repräsentiert eine vertikale oder horizontale Bildlaufleiste. Enthält die Werte des Positionszeigers für die Bildlaufleiste.
scroller.h	TScroller	Implementiert automatisches Fensterblättern.
signatur.h		Definiert die Kopf-Templates zum Extrahieren von Botschaften für ObjectWindows Ereignisbehandlungsfunktionen.
slider.h	TSlider THSlider TVSlider	Definiert das grundlegende Verhalten von Gleitreglern. Ein horizontaler Gleitregler. Ein vertikaler Gleitregler.
static.h	TStatic	Erzeugt ein statisches Dialogelement in einem Fenster.
statusba.h	TStatusBar	Konstruiert eine Statuszeile.
stgdoc.h	TStorageDocument	Unterstützt Operationen mit Verbunddatei-Strukturen.
textgadg.h	TTextGadget	Konstruiert ein Text-Gadgetobjekt.
tinycapt.h	TTinyCaption	Produziert eine kleinere Titelleiste für ein Fenster.
toolbox.h	TToolBox	Erzeugt ein Werkzeugpalettenobjekt mit der angegebenen Zahl von Zeilen und Spalten.
uihandle.h	TUIHandle	Definiert und zeichnet UI-Handles.
validate.h	TValidator, TPXPictureValidator, TFilterValidator, TRangeValidator, TLookupValidator, TStringLookupValidator	Basisprüfungsklasse, Bildprüfungsobjekt, Filterprüfungsobjekt, Bereichprüfungsobjekt, Nachschlageprüfung, String-Prüfung.
vbxctl.h	TVbxControl TVbxEventHandler TBIVbxLibrary	Schnittstelle für VBX-Dialogelemente. Behandelt Ereignisse von VBX-Dialogelementen. Lädt und initialisiert BIVBXxx.DLL.
version.h		Definiert die interne Versionsnummer der ObjectWindows-Bibliothek.
window.h	TWindow	Bietet fensterspezifisches Verhalten und kapselt viele Windows-API-Funktionen.
windowev.h		Definiert Ereignisbehandlungen und Antworttabellenmakros für Windows-Botschaften.

Die ObjectWindows-Ressourcendateien

Die ObjectWindows-Ressourcendateien definieren Ressourcen- und Kommando-IDs.

Tabelle 1.9: Übersicht über Ressourcendateien

Dateiname	Verwendung
Verzeichnis	OWL\INCLUDE
docview.rh	Definiert Ressourcen- und Kommando-IDs für docview.h und docview.rc.
edit.rh	Definiert Kommando-IDs für edit.h.
editfile.rh	Definiert Ressourcen- und Kommando-IDs für editfile.rc und editfile.h.
editsear.rh	Definiert Ressourcen- und Kommando-IDs für editsear.rc und editsear.h.
except.rh	Definiert String Ressourcen-IDs für except.h und except.rc.
inputdia.rh	Definiert Ressourcen-IDs für inputdia.rc und inputdia.h.
listview.rh	Definiert Ressourcen- und Befehls-IDs für listview.h
lkocale.rh	Definiert Lokalisierungsressourcen-IDs.
mdi.rh	Definiert Ressourcen- & Kommando-IDs für mdi.h.
oleview.rh	Definiert Ressourcen-IDs für die Verwendung mit Ole-fähigen Views.
printer.rh	Definiert Ressourcen-IDs für printer.rc und printer.h.
slider.rh	Definiert Ressourcen-IDs für slider.h.
statusba.rh	Definiert Ressourcen-IDs für statusba.h
validate.rh	Definiert Ressourcen für TValidator und abgeleitete Klassen.
window.rh	Definiert Kommando-IDs für window.h.

ObjectWindows Datentypen

ObjectWindows Datentypen sind aktualisiert worden,um portablere Typenbezeichner zu verwenden. Die folgende Tabelle enthält die Windows API Typbezeichner, die zugrundeliegenden C Typdefinitionen und die neuen ObjectWindows Typbezeichner. Um sicherzustellen, daß diese neuen Typen richtig verwendet werden, binden Sie die ObjectWindows Header-Dateien vor allen Windows Dateien (wie z.B. windows.h) in Ihren Anwendungsdateien ein. Der neue C++ Typ, welcher einen von Null verschiedenen Wert der Konstante **TRUE** zuweist, erlaubt Ihnen die Zuordnung ganzzahliger Werte zu **bool**schen Typen. Sie könne dann diese **bool**schen Werte mit **TRUE** vergleichen.

Tabelle 1.10: Neue ObjectWindows Datentypen

Windows	C Definition	Neuer Typ	Beschreibung
	char	int8	wird benutzt, wenn ein 8-bit-Wert mit Vorzeichen gefordert ist
BYTE	unsigned char	uint8	immer 8 Bits
WORD	unsigned short	uint16	immer 16 Bits
int	int	int	16 oder 32 Bits, je nach Plattform
UINT	unsigned int	uint	16 oder 32 Bits, je nach Plattform
LONG	long	long, int32	long-Wert (könnte 64 Bits auf einigen Plattformen bedeuten)

ULONG	unsigned long	ulong,int32	long-Wert (könnte 64 Bits auf einigen Plattformen bedeuten)
	long	int32	immer 32 Bits
DWORD	unsigned long	uint32	immer 32 Bits
BOOL	int	bool	neuer C++ Type, falls verfügbar; andernfalls mittels einer Aufzählung emuliert
TRUE	1	true	
FALSE	0	false	

K A P I T E L 2

ObjectWindows-Referenz

Dieses Kapitel listet die Klassen, Datenelemente, Elementfunktionen, Makros, Konstanten und Datentypen von ObjectWindows alphabetisch auf.

Die Header-Datei, die einen Eintrag definiert, steht jeweils dem Eintragsnamen gegenüber. Die Klassenelemente sind nach ihrem Zugriff-Spezifizierer gruppiert – public oder protected. Innerhalb dieser Kategorien werden die Datenelemente, dann die Konstruktoren (und der Destruktor, wenn einer existiert) und die Elementfunktionen alphabetisch aufgelistet.

Da viele Klasseneigenschaften in der Hierarchie von Basisklassen geerbt sind, werden nur Datenelemente und Elementfunktionen einer Klasse, die neu oder redefiniert sind, aufgelistet. Private Elemente werden nicht aufgeführt. Wenn Antworttabelleneinträge existieren, werden sie ebenso aufgelistet.

Für weitere Information über Windows-Typen wie zum Beispiel LPSTR siehe die Online-Hilfe.

Die kreuzreferenzierten Einträge bieten zusätzliche Informationen über die Verwendung des angegebenen Eintrages. Der erste Beispieleintrag verdeutlicht dieses Format.

Informationen über eine spezielle geerbte Elementfunktion können Sie über das Vererbungsdiagramm in Anhang C in Erfahrung bringen. Das Vererbungsdiagramm zeigt die Vorfahren der jeweiligen Klasse mit Ausnahme von TEventHandler und TStreamableBase, von denen alle Klassen geerbt haben. Ein Verzeichnis aller geerbten sowie aller überschriebenen virtuellen Funktionen finden Sie in der Online-Hilfe.

Klasse TBird [Beispiel] bird.h

Typdefinitionen

Dieser Abschnitt verzeichnet alle Typdefinitionen und Aufzählungstypen, die in einer Klasse definiert sind, in alphabetischer Reihenfolge.

unsigned short TOwlId Typdefinition

```
typedef unsigned short TOwlId;
```

Dieser Text erklärt den Inhalt dieserTypdefinition und wie Sie sie verwenden.

Siehe auch: Verwandte Datenelemente, Elementfunktionen, Klassen, Konstanten und Typen.

Öffentliche Datenelemente

Dieser Abschnitt listet alle öffentlichen Datenelemente und ihre Deklarationen alphabetisch auf und erläutert ihre Verwendung.

anOwlBeak

```
anOwlType anOwlBeak;
```

anOwlBeak ist ein Datenelement, das Informationen über diese Beispielklasse enthält. Dieser Text erklärt, was anOwlBeak enthält und wie es verwendet wird.

Siehe auch: Verwandte Datenelemente, Elementfunktionen, Klassen, Konstanten und Typen.

anOwlWing

```
anOwlType anOWLWing;
```

anOwlWing ist ein anderes öffentliches Datenelement.

Öffentliche Konstruktoren und Destruktor

Dieser Abschnitt listet alle öffentlichen Konstruktoren und den Destruktor für diese Klasse auf. Klassen können mehr als einen Konstruktor besitzen; sie haben jedoch niemals mehr als einen Destruktor.

Konstruktor

```
TBird(anOwlType aParameter);
```

Konstruktor für eine neue Beispielklasse; setzt das Datenelement anOwlBeak auf aParameter.

Destruktor

```
~TBird;
```

Destruktor für eine neue Beispielklasse; entfernt das Objekt TBird.

Öffentliche Elementfunktionen

Dieser Abschnitt listet alle öffentliche Elementfunktionen alphabetisch auf, die entweder für diese Klasse neu definiert sind oder die redefinierte vererbte Elementfunktionen sind. Wenn eine Funktion eine virtuelle Basisklassenfunktion überschreibt, wird dies im Text angegeben.

Das Schlüsselwort inline wird nicht aufgeführt, da es keinen Einfluß auf die Verwendung der Elementfunktionen hat.

EvGetDlgCode

```
uint OwlHoot();
```

Antwortet auf WM_GETDLGCODE-Botschaften.

OwlHoot

```
void OwlHoot();
```

Die Elementfunktion OwlHoot führt für die Beispielklasse Aktionen aus. Diese Funktion überschreibt die Funktion OwlHoot in ihrer Basisklasse TParent.

Siehe auch: TParent::OwlHoot

OwlSleep

```
virtual int OwlSleep(int index);
```

Die Funktion OwlSleep führt eine andere Aktion aus und überschreibt die Funktion OwlSleep in ihrer Basisklasse TParent.

Siehe auch: TParent::OwlSleep

Geschützte Datenelemente

Dieser Abschnitt listet alle geschützten Datenelemente und ihre Deklaration alphabetisch auf und erläutert ihre Verwendung.

anOwlFeather

```
anOwlType anOwlFeather;
```

anOwlFeather ist ein geschütztes Datenelement, das Informationen über diese Beispielklasse enthält.

Siehe auch: Verwandte Datenelemente, Elementfunktionen, Klassen, Konstanten und Typen.

Geschützte Konstruktoren

Konstruktor

```
TBird(anOwlType bParameter);
```

Wenn die Klasse einen geschützten Konstruktor besitzt, wird er hier aufgelistet.

Geschützte Elementfunktionen

Dieser Abschnitt listet alle geschützten Elementfunktionen auf.

OwlCry

BOOLEAN OwlCry;

Die Elementfunktion OwlCry führt für die Beispielklasse Aktionen aus.

Siehe auch: TSomethingElse::OwlCry

ZatsIt

```
virtual int ZatsIt(int index);
```

Die Funktion ZatsIt führt in der Klasse TBird eine spezielle Aktion aus.

Antworttabelleneinträge

Die Antworttabelle von TBird enthält dieses vordefinierte Makro für die EV_xxxx-Botschaften und ruft die folgende Elementfunktion auf:

Antworttabelleneintrag	Elementfunktion
EV_WM_GETDLGCODE	EVGetDlgCode

Schalter-Flag-Konstanten BF_xxxx — checkbox.h

Markierungsfeld- und Schaltfeldobjekte verwenden die Schalter-Flag-Konstanten, um den Zustand eines Auswahlfeldes anzuzeigen.

Tabelle 2.5: Schalter-Flag-Konstanten

Konstante	Bedeutung
BF_CHECKED	Element ist markiert.
BF_GRAYED	Element ist grau dargestellt.
BF_UNCHECKED	Element ist nicht markiert.

Siehe auch: TCheckbox::GetCheck, TCheckbox::SetCheck

Schalterbotschaftskonstanten BF_xxxx — windows.h

Mausobjekte und Aktionsschalter-Objekte verwenden die Schalterbotschaftskonstanten, um den Zustand eines Schalters anzuzeigen.

Konstante	Bedeutung
BN_CLICKED	Botschaft, die gesendet wird, wenn der Benutzer einen Schalter anklickt.
BN_DISABLE	Botschaft, die gesendet wird, wenn ein Schalter deaktiviert wird.
BN_DOUBLECLICKED	Botschaft, die gesendet wird, wenn der Benutzer einen Doppelklick auf einen Schalter ausführt
BN_HILITE	Botschaft, die gesendet wird, wenn der Benutzer einen Schalter markiert.
BN_PAINT	Botschaft, die gesendet wird, wenn ein Schalter neu gezeichnet werden muß.
BN_UNHILITE	Botschaft, die gesendet wird, wenn die Markierung eines Schalters aufgehoben werden muß.

Siehe auch: TRadioButton::BNClicked

Kombinationsfenster-Botschafts-Konstanten CBN_xxxx — windows.h

Konstante	Bedeutung
CBN_CLOSEUP	Botschaft, die gesendet wird, wenn das Listenfeld eines Kombinationsfensters geschlossen worden ist.
CBN_DBLCLK	Botschaft, die gesendet wird, wenn der Benutzer im Kombinationsfenster einen Doppelklick auf einen Text-String ausführt
CBN_DROPDOWN	Botschaft, die gesendet wird, wenn das Listenfeld eines Kombinationsfensters aufgeklappt wird.
CBN_EDITCHANGE	Botschaft, die gesendet wird, wenn der Benutzer den Text im Bearbeitungsteil eines Kombinationsfensters ändert.
CBN_EDITUPDATE	Botschaft, die gesendet wird, wenn der bearbeitete Text zur Anzeige ansteht.
CBN_ERRSPACE	Botschaft, die gesendet wird, wenn das Kombinationsfenster zu wenig Arbeitspeicher hat.
CBN_KILLFOCUS	Botschaft, die gesendet wird, wenn das Kombinationsfenster den Eingabefokus verliert.
CBN_SELENDCANCEL	Botschaft, die gesendet wird, wenn die ursprüngliche Auswahl des Benutzers aufgehoben werden muß, weil ein anderes Dialogelement-angewählt oder das Kombinationsfenster geschlossen wurde.
CBN_SELENDOK	Botschaft, die gesendet wird, wenn die Auswahl des Benutzers gültig ist.
CBN_SETFOCUS	Botschaft, die gesendet wird, wenn das Kombinationsfenster den Eingabefokus erhält.

Siehe auch: TComboBox

Editkonstanten CM_xxxx — window.rh

TEdit definiert auf Kommandos basierende Elementfunktionen, die als Antwort auf eine spezielle Auswahl oder Kommando des Menüs Bearbeiten aufgerufen werden.

Tabelle 2.6: Kommandobasierte Konstanten

Konstante	Elementfunktion	Menü-Äquivalent
CM_EDITCLEAR	TEdit::CMEditClear	Bearbeiten | Löschen
CM_EDITCOPY	TEdit::CMEditCopy	Bearbeiten | Kopieren
CM_EDITCUT	TEdit::CMEditCut	Bearbeiten | Ausschneiden
CM_EDITDELETE	TEdit::CMEditDelete	Bearbeiten | Löschen
CM_EDITPASTE	TEdit::CMEditPaste	Bearbeiten | Einfügen
CM_EDITUNDO	TEdit::CMEditUndo	Bearbeiten | Rückgängig
CM_EXIT	TWindow::CmExit	Datei/Schließen

Siehe auch: TEdit::CMEditClear, TEdit::CMEditCopy, TEdit::CMEditCut, TEdit::CMEditDelete, TEdit::CMEditPaste, TEdit::CMEditUndo, TWindow::CmExit

Editierdateikonstanten CM_xxxx docview.rh

Diese auf Kommandos basierenden Elementfunktionen werden als Antwort auf die Kommandos Öffnen, Schließen, Drucken und Speichern aufgerufen.

Tabelle 2.7: Kommandobasierte Konstanten

Konstante	Elementfunktion	Menü-Äquivalent
CM_FILECLOSE		Datei/Schließen
CM_FILENEW	TEditFile::CmFileNew	Datei | Neu
CM_FILEOPEN	TEditFile::CmFileOpen	Datei | Öffnen
CM_FILEPRINT		Datei | Drucken
CM_FILEPRINTERSETUP		Datei | Druckereinrichtung
CM_FILEREVERT	TDocManager::CmFileRevert	Datei/Wiederherstellen
CM_FILESAVE	TEditFile::CmFileSave	Datei | Speichern
CM_FILESAVEAS	TEditFile::CmFileSaveAs	Datei | Speichern unter
CM_VIEWCREATE	TDocManager::CmViewCreate	Datei/Ansicht erzeugen

Siehe auch: TEditFile::CmFileNew, TEditFile::CmFileOpen, TDocManager::CmFileRevert, TEditFile::CmFileSave, TEditFile::CmFileSaveAs

Bearbeiten-Ersetzen-Konstanten CM_xxxx editsear.rh

Diese auf Kommandos basierenden Elementfunktionen werden aufgerufen, wenn das korrespondierende Suchen- und Ersetzen-Kommando empfangen wurde.

Tabelle 2.9: Kommandobasierte Konstanten

Konstante	Elementfunktion	Menü-Äquivalent
CM_EDITFIND	TEditSearch::CMEditFind	Bearbeiten | Suchen
CM_EDITFINDNEXT	TEditSearch::CMEditFindNext	Bearbeiten | Weitersuchen
CM_EDITREPLACE	TEditSearch::CMEditReplace	Bearbeiten | Ersetzen

Siehe auch: TEditSearch::CMEditFind, TEditSearch::CMEditFindNext, TEditSearch::CMEditReplace

Edit View-Konstanten CM_xxxx oleview.rh

Diese auf Befehlen aufbauenden View-Funktionen werden als Antwort auf Menü- und Tastenkürzelbefehle aufgerufen. Die Auswahl Bearbeiten/Aktionsarten bezieht sich auf einen der OLE-spezifischen Menübefehle wie Bearbeiten oder Öffnen.

Konstante	Elementfunktion	Menü-Äquivalent
CM_EDITPASTESPECIAL	TOleWindow::CMEditPasteSpecial	Einfügen/Manuell
CM_EDITPASTELINK	TOleWindow::CMEditPasteLink	Einfügen/Link
CM_EDITINSERTOBJECT	TOleWindow::CMEditInsertObject	Einfügen/Objekt
CM_EDITLINKS	TOleWindow::CMEditLinks	Bearbeiten/Links
CM_EDITOBJECT	TOleWindow::CMEditObject	Bearbeiten/Objekt
CM_EDITFIRSTVERB	TOleWindow::EvCommandEnable	Bearbeiten/Aktionsarten
CM_EDITLASTVERB	TOleWindow::EvCommandEnable	Bearbeiten/Aktionsarten
CM_EDITCONVERT	TOleWindow::CMEdit	Bearbeiten/Umwandeln

MDI-Konstanten CM_xxxx mdi.rh

Diese MDI-Funktionen werden aufgerufen, wenn die korrespondierende MDI-Kommandobotschaft empfangen wurde.

Tabelle 2.10: Kommandobotschafts-konstanten

Konstante	Elementfunktion	Menü-Äquivalent
CM_ARRANGEICONS	TMDIClient::CmArrangeIcons	Fenster | Symbole anordnen
CM_CASCADECHILDREN	TMDIClient::CmCascadeChildren	Fenster | Überlappend
CM_CLOSECHILDREN	TMDIClient::CmCloseChildren	Fenster | Alle schließen
CM_CREATECHILD	TMDIClient::CmCreateChild	
CM_TILECHILDREN	TMDIClient::CmTileChildren	Fenster | Untereinander
CM_TILECHILDRENHORIZ	TMDIClient::CmTileChildren	Fenster | Nebeneinander

Makro DECLARE_RESPONSE_TABLE eventhan.h

Deklariert eine Antworttabelle in der Klassendefinition. Um Ereignisse für eine Klasse zu behandeln, müssen Sie eine Antworttabelle sowohl mit Hilfe dieses Makros innerhalb der Klasse deklarieren als auch die Antworttabelle mit Hilfe eines der DEFINE_RESPONSE_TABLE-Makros definieren. Verwenden Sie zum Beispiel für die Deklaration einer Antworttabelle

```
DECLARE_RESPONSE_TABLE(Class);
```

Dabei repräsentiert Class den Namen der aktuellen Klasse.

Die Antworttabellen von ObjectWindows definieren die Beziehung zwischen einer Fensterbotschaft und einer entsprechenden Ereignisbehandlungsfunktion. In der Klassenbeschreibung zu TEventHandler finden sich nähere Informationen über die Art und Weise, wie ObjectWindows einen Antworttabelleneintrag mit der passenden Funktion verbindet.

Siehe auch: Makro DEFINE_RESPONSE_TABLE,
Makro END_RESPONSE_TABLE, Klasse TEventHandler

Makro DEFINE_APP_DICTIONARY appdict.h

Dieses Makro definiert eine Referenz AppDictionary und ein Objekt AppDictionary, wie sie für die Verwendung in .DLL- und .EXE-Komponenten erforderlich sind. Sofern kein Benutzer-Dictionary angegeben wird, legt das Makro das OwlAppDictionary fest, ein global exportiertes TAppDictionary. Das Makro ist wie folgt definiert:

```
# define DEFINE_APP_DICTIONARY(AppDictionary)
```

Siehe auch: TAppDictionary

Makro DEFINE_DOC_TEMPLATE_CLASS docmanag.h

Dient dem Erzeugen eines Dokument-Template.
DEFINE_DOC_TEMPLATE_CLASS erwartet drei Argumente: den Namen der Dokumentklasse, die die Daten enthält, den Namen der View-Klasse, die die Daten darstellt, und den Namen der Template-Klasse. Die folgenden Beispiele aus DVSAMPLE.CPP, ein Beispielprogramm auf Ihrer Diskette, assoziieren Dokument- und View-Klassen mit neuen Template-Klassen.

```
DEFINE_DOC_TEMPLATE_CLASS(TFileDocument, TListView, ListTemplate);
DEFINE_DOC_TEMPLATE_CLASS(TFileDocument, TEditView, EditTemplate);
```

Siehe auch: TDocTemplate

Makros DEFINE_RESPONSE_TABLE eventhan.h

Das Makro DEFINE_RESPONSE_TABLEx erwartet eine Zahl-plus-x-Anzahl von Argumenten: den Namen der Klasse, die die Antworttabelle definiert und deren unmittelbare wie auch virtuelle Basisklassen.
Verwenden Sie das Makro END_RESPONSE_TABLE, um die Definition der Antworttabelle zu beenden. Zwischen den Makros DEFINE_RESPONSE_TABLE und END_RESPONSE_TABLE können Sie die Botschaftsantworteinträge für die Botschaften einfügen, die von der Klasse bearbeitet werden sollen. Ein Beispiel:

```
DEFINE_RESPONSE_TABLE1(TMyClass, TWindow)
  EV_WM_PAINT,
  EV_WM_LBUTTONDOWN,
END_RESPONSE_TABLE;
```

In diesem Beispiel verdeutlichen EV_WM_PAINT und EV_WM_LBUTTONDOWN die Botschaftsantworteinträge für die von TWindow abgeleitete Klasse TMyClass. Diese Makros rufen die entsprechenden Ereignisbehandlungsfunktionen EvPaint bzw. EvLButtonDown auf. Beachten Sie, daß gelegentlich auch Antworttabellen definiert sind, die keine Einträge besitzen. In solchen Fällen werden die Antworttabelleneinträge der Basisklassen nach der passenden Ereignisbehandlungsfunktion durchsucht. Sie können auch mehr als eine Botschaft mit einer Ereignisbehandlungsfunktion verbinden.

Die folgende Tabelle zeigt die Formen, die das Makro DEFINE_RESPONSE_TABLE in Abhängigkeit von der Zahl von Basisklassen annimmt.

Basisklassen	Makro
0	DEFINE_RESPONSE_TABLE(Class)
1	DEFINE_RESPONSE_TABLE(Class, Base)
2	DEFINE_RESPONSE_TABLE2(Class, Base1, Base2)
3	DEFINE_RESPONSE_TABLE3(Class, Base1, Base2, Base3)

Siehe auch: Makro DECLARE_RESPONSE_TABLE, Makro END_RESPONSE_TABLE, Klasse TEventHandler

Dialogelement-Botschaftskonstanten DLGC_xxxx windows.h

Zeigen an, welchen Typ Eingangsdaten der Dialogverwalter verarbeiten soll. Von EvGetDlgCode zurückgeliefert.

Konstante	Bedeutung
DLGC_BUTTON	Zeigt einen Schalter an.
DLGC_DEFPUSHBUTTON	Zeigt einen voreingestellten Schalter an.
DLGC_HASSETSEL	Zeigt einen Bereich von Zeichen in einem Bearbeitungselement an.
DLGC_RADIOBUTTON	Zeigt die Botschaft eines Aktionsschalters an.
DLGC_STATIC	Zeigt ein statisches Dialogelement an.
DLGC_UNDEFPUSHBUTTON	Zeigt einen nicht voreingestellten Aktionsschalter an.
DLGC_WANTALLKEYS	Kennzeichnet alle Tastatureingaben.
DLGC_WANTARROWS	Zeigt die Richtungstasten an.
DLGC_WANTCHARS	Kennzeichnet alle Tastenkürzelbotschaften.
DLGC_WANTMESSAGE	Zeigt an, daß alle Tastatureingaben an das Dialogelement weitergegeben werden können.
DLGC_WANTTAB	Kennzeichnet die Tabulatortaste.

Siehe auch: TButton::EvGetDlgCode, TCheckBox::EvGetDlgCode, TEdit::EvGetDlgCode, TListView::EvGetDlgCode, TSlider::EvGetDlgCode

Dokumentverwaltungs-Modus-Konstanten dmxxxx docmanag.h

TDocManager verwendet die dmxxxx-Konstanten, um anzuzeigen, ob ein Dokument nur einzelne oder auch mehrere offene Dokumente unterstützt und ob es Datei-Menü-IDs gibt.

Tabelle 2.11: Dokumentverwaltungs-Modus-Konstanten

Konstante	Bedeutung
dmMenu	Setzt IDs für das Datei-Menü.
dmMDI	Unterstützt mehrere offene Dokumente.
dmNoRevert	Deaktiviert das Kommando Datei | Zurücksetzen.
dmSaveEnable	Aktiviert das Menükommando Datei | Speichern.
dmSDI	Unterstützt nicht mehrere offene Dokumente.

Siehe auch: TDocManager::TDocManager

Dokumentbotschafts-Konstanten dnxxxx docmanag.h

TDocManager verwendet die dnxxxx-Botschaftskonstanten, um anzuzeigen, daß ein Dokument oder ein View erzeugt oder geschlossen wurde. Sie können über die Makros EV_OWLVIEW oder EV_OWLDOCUMENT Antworttabelleneinträge für diese Botschaften einrichten.

Tabelle 2.12: Dokumentbotschafts-aufzählung

Konstante	Bedeutung
dnCreate	Ein neues Dokument oder View wurde erzeugt.
dnClose	Ein Dokument oder View wurde geschlossen

Siehe auch: TDocManager::TDocManager

Dokument-Template-Konstanten dtxxxx locale.h

dtxxxx-Konstanten werden von TDocument und TDocTemplate verwendet, um Templates zu erzeugen. Mehrere Konstanten sind zu den von Windows in commdlg.h definierten OFN_xxxx-Konstanten äquivalent.

Tabelle 2.13: Dokument-View-Konstanten

Konstante	Windows-Äquivalent	Bedeutung
dtAutoDelete		Löscht das Dokument, wenn das letzte View gelöscht ist.
dtAutoOpen		Öffnet ein Dokument bei der Erzeugung.
dtCreatePrompt	(OFN_CREATEPROMPT)	Fragt den Benutzer vor Erzeugung eines Dokumentes, das aktuell nicht existiert.

dtDynRegInfo		Registriert einen Container oder Server für OLE 2-Unterstützung. Zeigt an, daß die Registrierinformationstabelle dynamisch nicht statisch ist.
dtFileMustExist	(OFN_FILEMUSTEXIST)	Erlaubt dem Benutzer nur die Eingabe existierender Dateinamen im Eingabefeld »Dateiname«. Wenn ein ungültiger Dateiname eingegeben wurde, wird eine Warnmeldung angezeigt.
dtHidden		Versteckt das Template vor der Auswahl durch den Benutzer.
dtHideReadOnly	(OFN_HIDEREADONLY)	Versteckt das Markierungsfeld für Lesezugriff.
dtNewDoc		Erzeugt ein neues Dokument mit nicht- angegebenem Pfad.
dtNoAutoView		Erzeugt den Standard-View-Typ nicht automatisch.
dtNoReadOnly	(OFN_NOREADONLYRETURN)	Gibt die angegebene Datei als beschreibbar zurück.
dtNoTestCreate	(OFN_NOTESTFILECREATE)	Führt keine Dokument-Erzeugungstests durch. Die Datei wird nach dem Schließen des Dialogfensters erzeugt. Wenn die Anwendung dieses Flag setzt, findet keine Überprüfung bezüglich Schreibschutz, einer vollen Diskette, einem geöffneten Laufwerksriegel oder eines Netzwerkschutzes dar. Für bestimmte Netzwerkumgebungen sollte dieses Flag gesetzt werden.
dtOverwritePrompt	(OFN_OVERWRITEPROMPT)	Wenn das »Speichern unter«-Dialogfenster gezeigt wird, wird der Benutzer gefragt, ob die Datei überschrieben werden kann.
dtPathMustExist	(OFN_PATHMUSTEXIST)	Erlaubt nur die Eingabe gültiger Dokumentpfade. Bei Eingabe eines ungültigen Pfadnamens wird eine Warnmeldung angezeigt.
dtProhibited	(OFN_ALLOWMULTISELECT) (OFN_ENABLEHOOK) (OFN_ENABLETEMPLATE) (OFN_ENABLETEMPLATEHANDLE)	Unterstützt diese angegebenen Windows-Optionen nicht.
dtReadOnly	(OFN_READONLY)	Überprüft das Markierungsfeld für Lesezugriff, wenn das Dialogfenster erzeugt wird.
dtRegisterExt		Wird verwendet, um einen Container oder Server für OLE 2-Unterstützung zu registrieren. Registriert bei dieser Anwendung eine Erweiterung.
dtSelected		Zeigt das zuletzt ausgewählte Template an.
dtSingleUse		Zeigt an, daß das Dokument für ausschließlichen Einzelbenutzerbetrieb registriert werden soll.
dtSingleView		Stellt nur ein einzelnes View für jedes Dokument bereit.
dtUpdateDir		Aktualisiert das Verzeichnis mit dem Dialogverzeichnis.

Siehe auch: TDocTemplate::GetFlags, TLocaleString, Klasse TDocument

Makro END_RESPONSE_TABLE — eventhan.h

```
END_RESPONSE_TABLE;
```

Dieses Makro zeigt das Ende der Antworttabelle an.

Siehe auch: Makro DEFINE_RESPONSE_TABLE

Editierbotschaftskonstanten EN_xxxx windows.h

Zeigen den Status eines Editierfelds in verschiedenen Situationen an: nachdem ein Benutzer Text geändert hat, wenn das Editierfeld den Eingabefokus erhalten hat, usw.

Konstante	Bedeutung
EN_CHANGE	Botschaft, die gesendet wird, wenn die Anzeige nach Änderungen im Editierfeld aktualisiert wird.
EN_ERRSPACE	Botschaft, die gesendet wird, wenn das Editierfeld über zu wenig Arbeitsspeicher verfügt
EN_HSCROLL	Botschaft, die gesendet wird, wenn der Benutzer die horizontale Bildlaufleiste anklickt.
EN_KILLFOCUS	Botschaft, die gesendet wird, wenn das Editierfeld den Eingabefokus verliert.
EN_MAXTEXT	Botschaft, die gesendet wird, wenn der einzufügende Text abgeschnitten wird.
EN_SETFOCUS	Botschaft, die gesendet wird, wenn das Editierfeld den Eingabefokus erhält.
EN_UPDATE	Botschaft, die gesendet wird, wenn das Editierfeld revidierten Text anzeigen wird.
EN_VSCHOLL	Botschaft, die gesendet wird, wenn der Benutzer die vertikale Bildlaufleiste anklickt.

Makros EV_xxxx windowev.h

Die EV_xxxx-Makros erzeugen Antworttabelleneinträge, die Ereignisse mit Elementfunktionen verbinden.

Tabelle 2.14: EV_xxxx-Makros

Makro	Bedeutung
EV_CHILD_NOTIFY(id,notifyCode,method)	Behandelt Benachrichtigungen mit der ID untergeordneter Fenster (z.B. Benachrichtigungsbotschaften von Schaltern, Editierfeldern, Listenfeldern, Kombinationsfenstern und Bildlaufleisten) beim übergeordneten Fenster. Übergibt keine Argumente.
EV_CHILD_NOTIFY_ALL_CODES	Übergibt alle Benachrichtigungen an die Antwortfunktion und übergibt den Benachrichtigungs-Code als Argument.
EV_CHILD_NOTIFY_AND_CODE(id, notifyCode, method)	Behandelt Benachrichtigungen mit der ID untergeordneter Fenster beim übergeordneten Fenster und übergibt den Benachrichtigungs-Code als Argument.
EV_COMMAND(id, method)	Behandlung für Menüauswahlen, Tastenkürzel und Aktionsschalter.
EV_COMMAND_AND_ID(id, method)	Behandlungsroutine für mehrere Kommandos, die eine einzelne Antwortfunktion verwenden. Übergibt die Menü-ID als Argument.

EV_COMMAND_ENABLE(id, method)	Aktiviert und deaktiviert Kommandos, wie zum Beispiel Schalter und Menüeinträge.
EV_MESSAGE(message, method)	Allgemeines Makro für Windows WM_xxxx-Botschaften.
EV_NOTIFY_AT_CHILD(notifyCode, method)	Behandelt alle Benachrichtigungen mit der ID untergeordneter Fenster beim untergeordneten Fenster.
EV_OWLDOCUMENT(id, method)	Behandelt Benachrichtigungen für neue Dokumente.
EV_OWLNOTIFY(id, method)	Generische Dokumentbehandlung.
EV_OWLVIEW(id, method)	Behandelt View-Benachrichtigungen.
EV_REGISTERED(str, method)	Behandelt registrierte MSG-Botschaften.
EV_VIEWNOTIFY	Sendet eine Benachrichtigungsbotschaft von den Dokumenten an die Views.

Funktion GetApplicationObject appdict.h

```
TApplication* GetApplicationObject(unsigned pid = 0);
```

Diese globale Funktion wurde hauptsächlich zur Sicherstellung der Abwärtskompatibilität mit früheren ObjectWindows-Anwendungen aufgenommen. GetApplicationObject ruft TAppDictionary::GetApplication für ein Anwendungs-Dictionary auf, um das mit einer Prozeßkennung verbundene Anwendungobjekt zu finden. Wenn GetApplicationObject von .EXE- oder .DLL-Dateien verwendet wird, die ObjectWindows statisch einbinden, kann die Funktion 0 zurückgeben.

Siehe auch: Funktion GetWindowPtr, TAppDictionary::GetApplication

Funktion GetWindowPtr window.h

```
TWindow* GetWindowPtr(HWND hWnd);
```

Diese globale Funktion wurde hauptsächlich zur Sicherstellung der Abwärtskompatibilität mit früheren ObjectWindows-Anwendungen aufgenommen. GetWindowPtr ruft zuerst die globale Funktion GetApplicationObject auf, um die Anwendung zu finden.Ruft danach GetWindowPtr von TApplication auf, um den Zeiger TWindow abzufragen, der mit dem Fenster verbunden ist.

Siehe auch: Funktion GetApplicationObject, TApplication::GetWindowPtr

Dateikonstanten ID_xxxx inputdia.rh

Dies sind die Ressourcen- und Dialogelement-IDs für das Eingabedialogfenster.

Tabelle 2.15: ID-Dateikonstanten

Konstante	Bedeutung
IDD_INPUTDIALOG	Ressourcen-ID-Nummer für das Eingabedialogfenster.
ID_INPUT	Dialogelement-ID für die Benutzereingabe.
ID_PROMPT	Dialogelement-ID für den statischen Text.

Siehe auch: TInputDialog::SetUpWindow

Druckerkonstanten ID_xxxx printer.rh

Die ID_xxxx- Druckerkonstanten sind die Ressourcen- und Dialogelement-IDs für das Dialogfenster zum Druckabbruch.

Tabelle 2.16: ID- Druckerkonstanten

Konstante	Bedeutung
IDD_ABORTDIALOG	Ressourcen-ID-Nummer für das Abbruch-Dialogfenster.
ID_TITLE	Dialogelement-ID für den ausgewählten Druckertreiber.
ID_DEVICE	Dialogelement-ID für den ausgewählten Drucker.
ID_PORT	Dialogelement-ID für den ausgewählten Druckerport.

Tastenkürzel-ID-Konstanten IDA_xxxx editfile.rh

IDA_EDITFILE ist eine Ressourcen-ID für Tastenkürzel.

Tabelle 2.17: Tastenkürzel-ID-Konstanten

Konstante	Bedeutung
IDA_EDITFILE	Ressourcen-ID für Tastenkürzel

OLE-Tastenkürzel-ID-Konstanten IDA_xxxx oleview.rh

Ressourcen-ID für Tastenkürzel.

Konstante	Bedeutung
IDA_OLEVIEW	Ressourcen-ID für Tastenkürzel für OLE-fähige Anwendungen.

Menü-ID-Konstante IDM_xxxx editfile.rh

IDM_EDITFILE ist eine Ressourcen-ID für Menüauswahlen.

Tabelle 2.18: Menü-ID-Konstanten

Konstante	Bedeutung
IDM_EDITFILE	Ressourcen-ID für Menüauswahlen.

OLE-Menü-ID-Konstante IDM_xxxx oleview.rh

Menü-IDs für OLE-fähige Anwendungen.

Konstante	Bedeutung
IDM_OLEPOPUP	Popup-Menü für OLE-fähige Anwendungen.
IDM_OLEVIEW	Auswahlmenü-Views für OLE-fähige Anwendungen

Edit-View-ID-Konstanten IDS_xxxx oleview.rh

String-Konstanten, die als Antwort auf Edit-View-Befehle verwendet werden

Konstante	Bedeutung
IDS_EDITOBJECT	Das Objekt bearbeiten
IDS_EDITCONVERT	Das Objekt konvertieren
IDS_EXITSERVER	Die Server-Anwendung verlassen

Modus-Konstanten IDS_xxxx statusba.rh

Ressourcen- und Befehls-IDs zur Verwendung mit TStatusBar.

Konstante	Bedeutung
IDS_MODES	String-Ressource für Bezeichner für Modus »ein«.
IDS_MODESOFF	String-Ressource für Bezeichner für Modus »aus«.

Dokument-String-ID-Konstanten IDS_xxxx docview.rh

Tabelle 2.19: Dokument-String-ID Konstanten

Konstante	Zeigt diese Meldungen an:
IDS_DOCCHANGED	Wenn das Dokument verändert wurde, wird folgende Meldung angezeigt: »Wollen Sie die Änderungen speichern?«
IDS_DOCLIST	Dokument ist ein Dokumenttyp.
IDS_DOCMANAGERFILE	Dies ist eine Dokumentverwaltungsdatei.
IDS_DUPLICATEDOC	Dies ist ein Zweitdokument.
IDS_NODOCMANAGER	Es ist keine Dokumentverwaltung vorhanden.
IDS_NOMEMORYFORVIEW	Der Arbeitsspeicher reicht nicht aus, um das Dokument anzuzeigen.
IDS_NOTCHANGED	Das Dokument wurde nicht geändert.
IDS_READERROR	Fehler beim Lesen der Datei.
IDS_UNABLECLOSE	Dokumentverwaltung kann das Dokument nicht schließen.
IDS_UNABLEOPEN	Dokumentverwaltung kann das Dokument nicht öffnen.
IDS_UNTITLED	Dokument ist unbenannt.
IDS_VIEWLIST	Dokument ist ein View-Typ.

Editierdatei-ID-Konstanten IDS_xxxx editfile.rh

ObjectWindows definiert diejenigen String-Konstanten, die von Edit- und Dateiklassen verwendet werden, um Informationen über Dateien anzuzeigen.

Tabelle 2.20: Editierdatei-ID-Konstanten

Konstante	Bedeutung
IDS_FILECHANGED	Der Text in der Datei wurde verändert. Wollen Sie die Änderungen speichern?
IDS_FILEFILTER	Verwenden Sie diesen Filter, um Textdateien darzustellen.
IDS_UNABLEREAD	Datei kann nicht von der Diskette gelesen werden.
IDS_UNABLEWRITE	Datei kann nicht auf Diskette geschrieben werden.
IDS_UNTITLEDFILE	Standard-Fenstertitel, sofern die Datei nicht bearbeitet wird.

Exception-Botschaften IDS_xxxx except.rh

Die folgende Liste umfaßt allgemeine und anwendungsbezogene Exception-Botschaftskonstanten, die nach ihrem Botschaftstyp gruppiert sind.

Tabelle 2.21: Exception-Botschaftskonstanten

Konstante	Bedeutung
IDS_INVALIDMAINWINDOW	Ungültiges Hauptfenster
IDS_INVALIDMODULE	Ungültiges Modul für das Fenster

IDS_NOAPP	Kein Anwendungsobjekt
IDS_OKTORESUME	Trotz Fehlers fortfahren
IDS_OWLEXCEPTION	Unbekannte Exception
IDS_OUTOFMEMORY	Speicher voll
IDS_UNHANDLEDXMSG	Nicht behandelter Fehler xmsg
IDS_UNKNOWNERROR	Unbekannter Fehler
IDS_UNKNOWNEXCEPTION	Unbekannter Exception-Fehler
Owl 1 Kompatibilitäts-Botschaften:	
IDS_INVALIDCHILDWINDOW	Ungültiges untergeordnetes Fenster
IDS_INVALIDCLIENTWINDOW	Ungültiges Client-Fenster
IDS_INVALIDWINDOW	Ungültiges Fenster
TXWindow Botschaften:	
IDS_CHILDCREATEFAIL	Erzeugungsfehler für ein untergeordnetes Fenster
IDS_CHILDREGISTERFAIL	Klassenregistrierungsfehler für ein untergeordnetes Fenster
IDS_CLASSREGISTERFAIL	Klassenregistrierungsfehler für ein Fenster
IDS_LAYOUTCOMPLETE	Layoutfenster-Fehler
IDS_LAYOUTBADRELWIN	Layoutfenster-Fehler
IDS_MENUFAILURE	Fehler bei der Menüerzeugung
IDS_PRINTERERROR	Druckerfehler
IDS_VALIDATORSYNTAX	Syntaxfehler bei Eingabeüberprüfung
IDS_WINDOWCREATEFAIL	Erzeugungsfehler für ein Fenster
IDS_WINDOWEXECUTEFAIL	Ausführungsfehler für ein Fenster
GDI Botschaften:	
IDS_GDIALLOCFAIL	GDI-Reservierungsfehler
IDS_GDICREATEFAIL	GDI-Erzeugungsfehler
IDS_GDIDELETEFAIL	GDI-Fehler beim Objekt löschen
IDS_GDIDESTROYFAIL	GDI-Fehler beim Objekt entfernen
IDS_GDIFAILURE	GDI-Fehler
IDS_GDIFILEREADFAIL	GDI-Dateilesefehler
IDS_INVALIDDIBHANDLE	Ungültiges DIB-Handle
IDS_GDIRESLOADFAIL	GDI-Ressourcen-Ladefehler

List-View-ID-Konstanten IDS_xxxx listview.rh

ObjectWindows verwendet List-View-Konstanten, um Operationen zu definieren, die auf Views ausgeführt werden. Dies schließt das Leeren des Dokumentes, das Einfügen einer neuen Zeile, das Kopieren von Text in die Zwischenablage usw. mit ein.

Tabelle 2.22: List-View String-ID-Konstanten

Konstante	Bedeutung
IDS_LISTVIEW	Ressourcen-ID für List-View-Konstanten

Drucker-String-ID-Konstanten IDS_xxxx printer.rh

ObjectWindows definiert mehrere Konstanten für den Drucker, um den Druckerstatus zu bestimmen.

Tabelle 2.23: Drucker-String-ID-Konstanten

Konstante	Angezeigter String
IDS_PRNCANCEL	Druck abgebrochen.
IDS_PRNERRORCAPTION	Druckerfehler.
IDS_PRNERRORTEMPLATE	Dokument nicht gedruckt.
IDS_PRNGENERROR	Fehler beim Drucken.
IDS_PRNMGRABORT	Druck in der Druckverwaltung abgebrochen.
IDS_PRNON	Drucker angeschaltet.
IDS_PRNOUTOFDISK	Plattenspeicher voll.
IDS_PRNOUTOFMEMORY	Speicher voll.

Prüfungs-ID-Konstanten IDS_xxxx validator.rh

ObjectWindows definiert mehrere von Prüfungsklassen verwendete Konstanten, um den Prüfungsstatus zu bestimmen.

Tabelle 2.24: Prüfungs-ID-Konstanten

Konstante	Bedeutung
IDS_VALPXPCONFORM	Element stimmt nicht mit dem korrekten Bildformat überein.
IDS_VALINVALIDCHAR	Zeichen ist kein gültiger Eintrag.
IDS_VALNOTINRANGE	Eintrag liegt nicht im angegebenen Bereich.
IDS_VALNOTINLIST	String steht nicht in der Liste gültiger Einträge.

Konstante IDW_MDICLIENT framewin.h

```
IDW_MDICLIENT
```

Konstante einer untergeordneten Fenster-ID für die Identifizierung von MDI-Client-Fenstern.

Konstante IDW_MDIFIRSTCHILD framewin.h

```
IDW_FIRSTMDICHILD
```

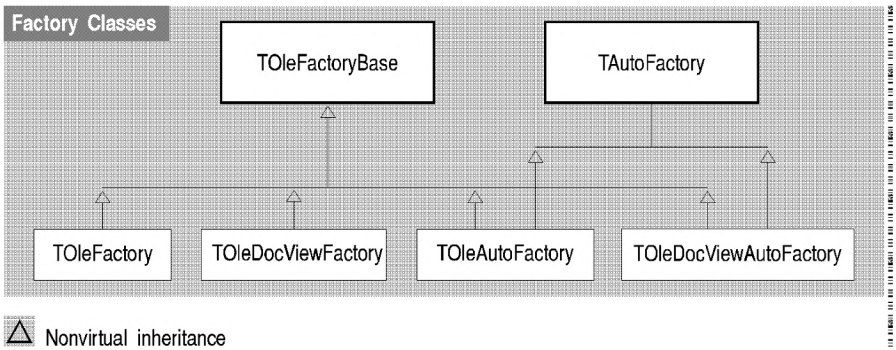
Konstante einer untergeordneten Fenster-ID für die Identifizierung des ersten MDI-Client-Fensters.

Klassen-Templates Factory olefacto.h

Die Klassen-Templates Factory erzeugen Callback-Code für Dok/View- und Nicht-Dok/View-ObjectWindows-Anwendungen, und zwar ebenso für automatisierte wie für nicht automatisierte Anwendungen. Verwenden Sie die Factory-Klassen-Templates, um Objekte für das Verknüpfen und Einbetten herzustellen. (Wenn ein Anwendungsobjekt in einen Container eingebettet werden muß, ist die Callback-Funktion für die Erzeugung des eingebetteten Objekts verantwortlich.) Je nachdem, mit welchen Argumenten Sie die »Schablone« der Factory-Klasse ausstatten, erhalten Sie eine jeweils andere Art von »Fabrik«, die gerade für die Herstellung des Objekts ausgelegt ist, das Sie benötigen. Alle »Schablonen«-Klassen gehen jedoch davon aus, daß die Klassen ein globales AppDictionary verwenden (das Dictionary der Anwendung) sowie einen globalen Registrar (den Zeiger TOcRegistrar, der die Registrierung der Anwendung in der Datenbank verwaltet).

ObjectWindows enthält verschiedene Factory-Klassen-Templates, die zwei Hauptkategorien angehören: solche, die für Dok/View-Anwendungen vorgesehen sind und solche für Nicht-Dok/View-Anwendungen.Obwohl all diese Klassen die gleichen Funktionen enthalten, sind sie für die Erzeugung unterschiedlicher Objekttypen ausgelegt.

Das Hierarchiediagramm veranschaulicht die Vererbungsrelation unter diesen Klassen.



Jede Klasse nimmt drei Argumente an: die Anwendungsklasse, die Automatisierungsklasse und die Dok/View-Klasse. Die Argumente zeigen an, ob die Anwendung eine Dok/View-Anwendung ist und ob sie automatisiert ist oder nicht. Die Factory-Klassen und ihre Definitionen schließen die folgenden vier Klassen ein

Factory-Klasse für nicht-automatisierte Dok/View-OLE-Komponenten

```
template <class T> class TOleDocViewFactory
```

```
: public TOleFactoryBase<T, TOleFactoryDocView<T,
  TOleFactoryNoAuto<T>>{};
```

Factory-Klasse für automatisierte Dok/View-OLE-Komponenten

```
template <class T> class TOleDocViewAutoFactory

: public TOleFactoryBase<T, TOleFactoryDocView<T,
  TOleFactoryAuto<T>>{};
```

Factory-Klasse für nicht automatisierte Nicht-Dok/View-OLE-Komponenten

```
template <class T> class TOleFactory

: public TOleFactoryBase<T, TOleFactoryNoDocView<T,
  TOleFactoryNoAuto<T>>{};
```

Factory-Klasse für automatisierte Nicht-Dok/View-OLE-Komponenten

```
template <class T> class TOleAutoFactory

: public TOleFactoryBase<T, TOleFactoryNoDocView<T,
  TOleFactoryAuto<T>>{};
```

Gleichgültig, ob es sich um eine Dok/View- oder um eine Nicht-Dok/View-Anwendung handelt, muß sie bei Ihrer Funktion OwlMain registriert werden. Das an TOcRegistrar übergebene Argument (in diesem Fall TOleFactory<>TDrawApp>>) konstruiert ein Objekt und wandelt es in einem Typ TComponentFactory um. Es verwendet dazu den Operator TComponentFactory, um einen Funktionszeiger zu erzeugen. Tatsächlich wird das Objekt aber nicht real erzeugt, weil alle Funktionen der Factory-Klasse statisch sind.

Übergeben Sie Ihr aus TApplication abgeleitetes Anwendungsobjekt als Parameter an TOleFactory, wie im folgenden Code aus STEP15.CPP demonstriert:

```
int

OwlMain(int /*argc*/, char* /*argv*/ [])

{

  try {

    Registrar = new TOcRegistrar(AppReg, TOleFactory
     awApp(),

                                  TApplication::GetCmdLine());

    if (Registrar-IsOptionSet(TOcCmdLine::AnyRegOption))

      return 0;

//Wenn es sich hier um einen normalen .EXE-Server handelt, führen Sie
//die Anwendung jetzt aus; warten Sie andernfalls bis zum Aufruf der
Factory..

    return Registrar-Run();

}
```

Im allgemeinen führt jede Factory-Klasse im Standard-Callback Code die folgenden Schritte aus:

Die Factory erhält die Anwendung. Dies ist das aus TApplication abgeleitete Anwendungsobjekt. Im Falle eines DLL-Servers kann es mehrere Instanzen des Objekts geben. Die Factory überprüft mittels TAppDictionary::GetApplication, ob es im Anwendungs-Dictionary einen Eintrag für ein Anwendungsobjekt für den aktuellen Prozeß gibt.

Wenn die Anwendung nicht existiert, erzeugt die Factory das Anwendungsobjekt. Dann überprüft sie zunächst, ob das Erzeugen der Anwendung erfolgreich war, bevor sie das ihr entsprechende Objekt TOcApp erzeugt. Ist das Options-Flag shutdown gesetzt, beendet sich die Factory anschließend. (Wenn die Anwendung bereits entfernt wurde, ist das Flag shutdown gesetzt).

Wenn der Factory ein Options-Flag shutdown übergeben wird (einer der Werte des Aufzählungstyps TOcAppMode), schließt sie die Anwendung und ruft die Funktion DestroyApp der Factory auf, um die Anwendung zu entfernen.

Wenn es sich um eine automatisierte Anwendung handelt, erzeugt die Factoy das entsprechende Automatisierungsobjekt.

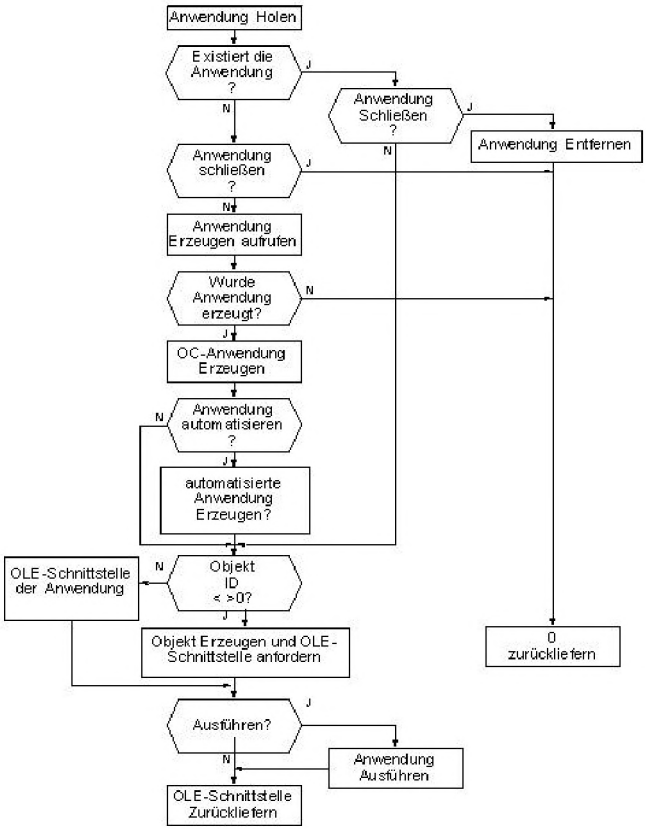
Wenn die Objekt-ID ungleich 0 ist, erzeugt die Factory das Objekt und holt die OLE-Schnittstelle. Andernfalls holt sie die OLE-Schnittstelle der Anwendung. Die Dok/View- und Nicht-Dok/View-Prozesse weichen in diesem Punkt voneinander ab, weil sie unterschiedliche Objekttypen erzeugen müssen. Ist die Anwendung automatisiert, erzeugt die Factory ein automatisiertes Helferobjekt.

Die Factory überprüft, ob das Options-Flag amRun (einer der Werte des Aufzählungstyps TOcAppMode) gesetzt ist, und führt, wenn das zutrifft, die Anwendung aus. Dies ist dann der Fall, wenn die Anwendung entweder als .EXE compiliert wurde oder eine DLL ist, die als .EXE ausgeführt wird. Wenn das Flag amRun nicht gesetzt ist, und die Anwendung ein In-Proc-DLL-Server ist, der nicht ausgeführt werden soll, startet die Factory die Anwendung lediglich.

Die Factory liefert entweder die OLE-Schnittstelle für das Objekt zurück oder 0, wenn keine Schnittstelle angefordert wurde oder ein Fehler aufgetreten ist.

Das folgende Diagramm verdeutlicht dieses Vorgehen.

Die Factory kann zurückgerufen werden und diesen Prozeß solcherart mehrfach ausführen:



Beim ersten Callback erzeugt die Factory die Anwendung. Ist das Flag amRun gesetzt. tritt diese in die Botschaftsschleife ein.

Beim zweiten Callback ruft OLE die Factory auf, um ein Objekt zu automatisieren, es einzubetten oder zu verknüpfen. Im Falle eines eingebetteten und/oder verknüpften Objekts kann dieser Schritt einmal oder mehrmals durchgeführt werde. (Jedoch nicht bei automatisierten Objekten; hier kann dieser zweite Schritt stets nur einmal ausgeführt werden, weil alle folgenden Anforderungen die automatisierte Anwendung selbst durchlaufen.)

Beim abschließenden Callback schließt die Factory die Anwendung.

Siehe auch: Klasse TAutoFactory, TAutoFactory::DestroyApp, Typdefinition TComponentFactory, Aufzählungstyp TOcAppMode, Klasse TOcRegistrar, Klasse TOleFactoryBase, TOleFactoryBase::DestroyApp, TOleFactory::TComponentFactory

ID-Konstanten LangXxxx locale.h

Eine Sprachkennung ist eine 16-Bit-Zahl, die eine Sprache repräsentiert. In ObjectComponents ist eine Sprachkennung ein Wert vom Typ TLangId.

Eine Sprachkennung besteht aus zwei Teilen. Die zehn niederwertigen Bits bezeichnen eine Sprache und die sechs hohen Bits bezeichnen einen Dialekt oder eine Variante. Beispielsweise kann in einer Sprachkennung einfach Französisch festgelegt werden oder mit Hilfe der oberen Bits in belgisches Französisch, schweizerisches Franmzösisch oder kanadisches Französisch unterschieden werden.

ObjectComponents definiert Konstanten, die drei allgemeine IDs darstellen:

Konstante	Bedeutung
LangSysDefault	Die für das System voreingestellte Sprache.
LangUserDefault	Die für einen bestimmten Benutzer voreingestellte Sprache (die bei Mehrbenutzersystemen von der Systemeinstellung abweichen kann).
LandNeutral	Eine neutrale Einstellung, die keine bestimmte Sprache bezeichnet. Eine Anwendung, die diesen Wert erhält, verwendet ihre eigene Voreinstellung.

Siehe auch: Typdefinition TLangId, TLocaleString

Listenfensterbotschafts-Konstanten LBN_xxxx windows.h

Zeigen Änderungen im Status des Listenfensters an.

Konstante	Bedeutung
LBN_DBLCLK	Botschaft, die gesendet wird, wenn der Benutzer einen Doppelklick auf einen String in einem Listenfenster ausführt.
LBN_ERRSPACE	Botschaft, die gesendet wird, wenn dem Listenfenster zu wenig Arbeitsspeicher zur Verfügung steht.
LBN_KILLFOCUS	Botschaft, die gesendet wird, wenn das Listenfenster den Eingabefokus verliert.
LBN_SELCANCEL	Botschaft, die gesendet wird, wenn der Benutzer die Auswahl in einem Listenfenster aufhebt.
LBN_SELCHANGE	Botschaft, die gesendet wird, wenn der Benutzer die Auswahl in einem Listenfenster ändert.
LBN_SETFOCUS	Botschaft, die gesendet wird, wenn das Listenfenster den Eingabefokus erhält.

Konstante lmParent — layoutco.h

```
#define lmParent 0
```

LmParent wird für das Konstruieren von Layoutmaßen verwendet (zum Beispiel für Rand- und Größenrestriktionen).

Siehe auch: TLayoutConstraint

Funktion LongMulDiv — scroller.h

```
inline long LongMulDiv(long mul1, long mul2, long div1);
```

TScroller verwendet diese Funktion, um horizontale Bereichswerte (XRange) von der Bildlaufleiste in horizontale Bildlaufwerte (XScrollValue) und umgekehrt zu konvertieren, oder um vertikale Bereichswerte (YRange) von der Bildlaufleiste in vertikale Bildlaufwerte (YScrollValue) und umgekehrt zu Konvertieren.

Siehe auch: TScroller

Konstante MAX_RSRC_ERROR_STRING — except.h

```
const int MAX_RSRC_ERROR_STRING = 255;
```

Maximale Zahl von Zeichen für eine Fehlerbotschaft.

Botschafts-Konstanten MB_Xxxx — windows.h

Enthält Informationen über Eigenschaften und Verhalten eines Meldungsdialogfensters. Sie können mit Hilfe dieser Konstanten festlegen, welche Art von Informationen in einem Meldungsdialogfenster angezeigt werden.

Konstante	Bedeutung
MB_OK	Das Meldungsdialogfenster enthält einen Aktionsschalter »OK«.
MB_OKCANCEL	Das Meldungsdialogfenster enthält Aktionsschalter »Abbrechen« und »OK«.,
MB_ABORTRETRYIGNORE	Das Meldungsdialogfenster enthält Aktionsschalter »Abbrechen«, »Wiederholen« und »Übergehen«.
MB_YESNOCANCEL	Das Meldungsdialogfenster enthält Aktionsschalter »Ja«, »Nein« und »Abbrechen«.
MB_YESNO	Das Meldungsdialogfenster enthält Aktionsschalter »Ja« und »Nein«.
MB_RETRYCANCEL	Das Meldungsdialogfenster enthält Aktionsschalter »Wiederholen«

	und »Abbrechen«.
MB_ICONHAND	Das Meldungsdialogfenster enthält ein Stoppschildsymbol.
MB_ICONQUESTION	Das Meldungsdialogfenster enthält ein Fragezeichen.
MB_ICONEXCLAMATION	Das Meldungsdialogfenster enthält ein Ausrufezeichen.
MB_ICONASTERISK	Das Meldungsdialogfenster enthält ein Symbol, das aus dem Kleinbuchstaben i besteht.
MB_ICONINFORMATION	Das Meldungsdialogfenster enthält ein Symbol, das aus dem Kleinbuchstaben i besteht.
MB_ICONSTOP	Das Meldungsdialogfenster enthält ein Stoppschildsymbol.
MB_ICONHAND	Das Meldungsdialogfenster enthält ein Stoppschildsymbol.
MB_DEFBUTTON1	Der erste Aktionsschalter ist der voreingestellte Aktionsschalter im Meldungsdialogfenster. Diese Grundeinstellung ist immer aktiv, wenn MB_DEFBUTTON2 oder MB_DEFBUTTON3 nicht definiert sind.
MB_DEFBUTTON2	Der zweite Aktionsschalter ist der voreingestellte Aktionsschalter im Meldungsdialogfenster.
MB_DEFBUTTON3	Der dritte Aktionsschalter ist der voreingestellte Aktionsschalter im Meldungsdialogfenster
MB_APPLMODAL	Der Benutzer kann die Arbeit in diesem Fenster erst nach einer Reaktion auf das Meldungsdialogfenster fortsetzen. Er kann jedoch in anderen Fenstern arbeiten. MB_APPLMODAL ist die Voreinstellung und aktiv, sofern nicht MB_SYSTEMMODAL, MB_TASKMODAL oder MB_NOFOCUS festgelegt werden.
MB_SYSTEMMODAL	Der Benutzer kann die Arbeit in diesem Fenster erst nach einer Reaktion auf das Meldungsdialogfenster fortsetzen. Wenn die Anwendung nicht MB_ICONHAND anzeigt, wird das Meldungsfenster erst nach seiner Erzeugung modal. Das besitzende Fenster und andere Fenster können weiterhin Botschaften empfangen. Sie können MB_SYSTEMMODAL verwenden, um den Benutzer auf schwerwiegende Fehler hinzuweisen, wie nicht ausreichenden Arbeitsspeicher für die Ausführung einer Anwendung, auf die sofort reagiert werden muß.
MB_TASKMODAL	Der Benutzer kann die Arbeit in diesem Fenster erst nach einer Reaktion auf das Meldungsdialogfenster fortsetzen. Er kann jedoch in anderen Fenstern arbeiten. Anders als bei MB_APPLMODAL sind alle Fenster der obersten Ebene der aktuellen Task deaktiviert. Verwenden Sie diese Konstante, wenn die aufrufende Anwendung nicht über ein Fenster-Handle verfügt und Sie Eingaben in andere Fenster der aktuellen Anwendung verhindern wollen, ohne zugleich die Ausführung der anderen Anwendungen zu verhindern.

Siehe auch: TDocument::PostError, TDocManager::PostDocError, TWindow::MessageBox

Funktion NBits — color.h

```
uint16 NBits(int colors);
```

Gibt die zur Farbzahl korrespondierende Bitzahl zurück.

Siehe auch: Klasse NColors, TColor

Funktion NColors color.h

```
int NColors(uint16 bitCount);
```

Gibt zur gegebenen Bitzahl die korrespondierende Farbzahl zurück oder ist -1, wenn die Bitzahl nicht von Windows unterstützt wird. Aktuell unterstützte Bitzahlen sind 1, 4, 8 und 24.

Siehe auch: Klassen NBits, TColor

Dokument-Öffnen-Aufzählung ofxxxx docview.h

Diese Aufzählung definiert die Dokument- und Öffnen-Zugriffsmodi für das Konstruieren von Streams und das Speichern von Daten. Alle Konstanten mit derselben Funktionalität wie die von OLE 2.0 Dokdateien verwendeten sind in der folgenden Tabelle angezeigt; zum Beispiel STGM_TRANSACTED, STGM_CONVERT, STGM_PRIORITY und STGM_DELETEONRELEASE.

Zwar werden üblicherweise Dateien zur Datenspeicherung verwendet, es können aber auch Datenbanken oder Arbeitsblätter verwendet werden. I/O-Streams greifen auf diese Bitwerte eher als DOS zurück. Dokumente öffnen das Objekt zur Speicherung in einem der folgenden Modi:

Konstante	Bedeutung
ofParent	Ein Speicherobjekt wurde unter Verwendung des Modus des übergeordneten Objektes geöffnet.
ofRead	Ein Speicherobjekt wurde zum Lesen geöffnet.
ofWrite	Ein Speicherobjekt wurde zum Schreiben geöffnet.
ofReadWrite	Ein Speicherobjekt wurde zum Lesen und Schreiben geöffnet.
ofAtEnd	Positioniere nach dem Öffnen ans Ende der Datei.
ofAppend	Daten werden an das Ende des Speicherobjektes angehängt.
ofTruncate	Eine bereits existierende Datei wird abgeschnitten.
ofNoCreate	Öffnen schlägt fehl, wenn Datei nicht existiert.
ofNoReplace	Öffnen schlägt fehl, wenn Datei bereits existiert.
ofBinary	Daten werden im binären, nicht im textuellen Format gespeichert. Zeilenrückläufe werden nicht entfernt.
ofIosMask	Alle oben stehende Bits werden durch die Klasse ios benutzt.
ofTransacted	Änderungen am Speicherobjekt werden behalten, bis die Daten entweder an den permanenten Speicher übergeben oder verworfen werden. (STGM_TRANSACTED)
ofPreserve	Legt eine Sicherung der vorherigen Speicherdatenverwendung an,

	bevor ein Speicherobjekt mit demselben Namen erzeugt wird. (STGM_CONVERT)
ofPriority	Unterstützt vorläufiges, effizientes Lesen, bevor der Speicher geöffnet wird. (STGM_PRIORITY)
ofTemporary	Der Speicher oder Stream wird automatisch entfernt, wenn das Objekt entfernt wird. (STGM_DELETEONRELEASE)

Siehe auch: TStream, TInStream, TOutStream

Eigenschaftsattributkonstanten pfxxxx docview.h

Diese Konstanten definieren Eigenschaftsattribute für Dokument und View. Dokumente, Views und Anwendungen verwenden diese Attribute, um die Verarbeitung eines Dokuments oder eines View zu bestimmen.

Konstante	Bedeutung
pfGetText	Eigenschaft ist in einem Textformat zugreifbar.
pfGetBinary	Eigenschaft ist im ursprünglichen Nicht-Textformat zugreifbar.
pfConstant	Eigenschaft kann für die Objektinstanz nicht verändert werden.
pfSettable	Eigenschaft kann als ursprüngliches Format gesetzt werden.
pfUnknown	Eigenschaft ist definiert, jedoch für das Objekt nicht verfügbar.
pfHidden	Eigenschaft sollte vor dem Benutzer während des normalen Durchlaufs versteckt werden.
pfUserDef	Eigenschaft wurde zur Laufzeit vom Benutzer definiert.

Siehe auch: TDocument, TView

Makro _BUILDOWLDLL owldefs.h

```
_BUILDOWLDLL
```

_BUILDOWLDLL muß für das Erzeugen der ObjectWindows-DLL definiert werden und wird intern für die Kontrolle der Werte für die Makros _OWLCLASS, _OWLDATA und _OWLFUNC verwendet. Es ist für das Erzeugen der ObjectWindows-DLL in ObjectWindows Make-Dateien eingebunden.

Siehe auch: _OWLDLL

Makro _OWLCLASS owldefs.h

```
_OWLCLASS
```

Wird intern von ObjectWindows verwendet. _OWLCLASS ist eine ObjectWindows-Version von _RTLCLASS, die an den Im- und Export von WIN32-DLLs angepaßt wurde.

Für statisch eingebundene WIN16- und WIN32-DLLs werden die Standardmodelle verwendet. Wenn ObjectWindows-Anwendungen erzeugt werden, ergibt sich der Wert dieses Makros sowohl für WIN16- als auch für WIN32-DLLs zu _export. Sonst ergibt sich das Makro für WIN32-DLLs zu _import und führt notwendige Operationen an WIN32-DLLs durch. Bei WIN16 ergibt sich überhaupt kein Wert für dieses Makro.

Makro _OWLDATA owldefs.h

```
_OWLDATA
```

Das Makro _OWLDATA ist die ObjectWindows-Version von _RTLDATA und ist dafür angepaßt, um WIN32-DLLs für ObjectWindows zu exportieren und zu importieren. _OWLDATA verändert eine bestimmte Datendeklaration.

Für statisch eingebundene WIN16- und WIN32-DLLs werden die Standardmodelle verwendet. Wenn ObjectWindows-Anwendungen erzeugt werden, ergibt sich der Wert dieses Makros sowohl für WIN16- als auch für WIN32-DLLs zu _export. Sonst ergibt sich das Makro für WIN32-DLLs zu _import, und führt notwendige Operationen an WIN32-DLLs durch. Bei WIN16 ergibt sich überhaupt kein Wert für dieses Makro.

Makro _OWLDLL owldefs.h

```
_OWLDLL
```

_OWLDLL wird automatisch definiert, wenn _RTLDLL angeschaltet ist, und kontrolliert Werte für die Makros _OWLCLASS, _OWLDATA und _OWLFUNC. Es wird automatisch definiert, wenn das _OWLDLL-Benutzermodul von einem anderen Benutzer als DLL verwendet wird. Es muß definiert werden, wenn Sie ObjectWindows-Anwendungen oder -DLLs schreiben, die auf DLLs zurückgreifen. Dieses Makro kann ebenso durch eine Make-Datei angeschaltet werden.

Makro _OWLFAR owldefs.h

```
_OWLFAR
```

Das Makro _OWLFAR ist die ObjectWindows-Version von _RTLFAR und ist angepaßt, um far-Datenzeiger in DLLs für ObjectWindows zu unterstützen.

Makro _OWLFARVTABLE owldefs.h

```
_OWLFARVTABLE
```

Verschiebt die Tabellen der virtuellen Funktionen (vtables) von ObjectWindows aus der DGROUP des Datensegments in das Code-Segment. Verwenden Sie dieses Makro in Verbindung mit dem Makro _OWLCLASS, um beim Compilieren statischer Modelle die Option -huge zu setzen.

Makro _OWLFASTTHIS owldefs.h

```
_OWLFASTTHIS
```

Mit dem Makro _OWLFASTTHIS wird -fastthis in das Makro _OWLCLASS aufgenommen. Das bewirkt, daß alle ObjectWindows-Klassen die Aufrufkonvention fastthis für die Übergabe des Parameters this in die Register verwenden. Dieses Makro bewirkt das gleiche wie die Compiler-Option -po und kann nur für 16-Bit-Code eingesetzt werden.

Makro _OWLFUNC owldefs.h

```
_OWLFUNC
```

Das Makro _OWLFUNC ist die ObjectWindows-Funktionsversion von _RTLFUNC und ist dafür angepaßt, um Funktionen für das Erzeugen von WIN32-DLLs für ObjectWindows zu exportieren und zu importieren.

_OWLFUNC modifiziert ein bestimmtes Element oder eine globale Funktion für die Verwendung in einer DLL.

Für statisch eingebundene WIN16- und WIN32-DLLs werden die Standardmodelle verwendet. Wenn ObjectWindows-Anwendungen erzeugt werden, ergibt sich der Wert dieses Makros sowohl für WIN16- als auch für WIN32-DLLs zu _export. Sonst ergibt sich das Makro für WIN32-DLLs zu _import, und führt notwendige Operationen an WIN32-DLLs durch. Bei WIN16 ergibt sich überhaupt kein Wert für dieses Makro.

Funktion OWLGetVersion owldefs.h

Elementfunktion

```
unsigned short far _OWLFUNC OWLGetVersion();
```

Gibt die Versionsnummer der ObjectWindows-Bibliothek zurück. Die Versionsnummer ist als ein unsigned short repräsentiert.

Bildlaufleisten-Konstanten SB_Xxxx windows.h

Die folgenden Konstanten legen Modi für Bildlaufleisten fest:

Konstante	Bedeutung
SB_BOTH	Vertikale und horizontale Bildlaufleisten für ein Fenster werden angezeigt oder verborgen.
SB_CTL	Das Steuerelement einer Bildlaufleiste wird angezeigt oder verborgen.
SB_HORIZ	Die horizontalen Bildlaufleisten eines Fensters werden angezeigt oder verborgen.
SB_VERT	Die vertikalen Bildlaufleisten eines Fensters werden angezeigt oder verborgen.

Aufzählung shxxxx Dokument-Sharing docview.h

Die folgenden Datei-Sharing-Modi sind verfügbar, wenn Dokument-Streams geöffnet werden.

Tabelle 2.25: shxxxx-Konstanten

Konstante	Bedeutung
shCompat	Verwendet für nichtverträgliche Anwendungen, sollte aber nach Möglichkeit vermieden werden.
shNone	DENY_ALL-Funktionalität.
shRead	DENY_WRITE-Funktionalität.
shWrite	DENY_READ-Funktionalität.
shReadWrite	DENY_NONE-Funktionalität.
shDefault	Verwenden Sie den Standardwert für Stream-Implementierung.
shMask	shCompat|shNone|shRead|shWrite

Typdefinition TActionFunc window.h

```
typedef void(*TActionFunc)(TWindow* win, void* param);
```

Übergibt einen Funktionszeiger an TWindow::ForEach.

Siehe auch: TWindow::ForEach

Typdefinition TActionMemFunc — window.h

```
typedef void(TWindow::*TActionMemFunc)(TWindow* win, void* param);
```

Übergibt einen Elementfunktionszeiger an TWindow::ForEach.

Siehe auch: TWindow::ForEach

Typdefinition TAnyPMF — dispatch.h

```
typedef void(GENERIC::*TAnyPMF)();
```

TAnyPMF ist ein generischer Zeiger auf eine Elementfunktion.

Typdefinition TAnyDispatcher — dispatch.h

```
typedef LRESULT(*TAnyDispatcher)(GENERIC&, TAnyPMF, WPARAM, LPARAM);
```

TAnyDispatcher ist ein Botschaftszuteilungsroutinen-Typ. Alle Funktionen für die Zuteilung von Botschaften stimmen mit diesem Typ überein und erwarten vier Parameter:

- eine Referenz auf ein Objekt
- einen Zeiger auf eine Elementfunktion, bei der der Kopf in Abhängigkeit des von der Funktion durchgeführten Extrahierens von Botschaften variiert.
- WPARAM
- LPARAM

Klasse TApplication — applicat.h

Als eine von TModule abgeleitete Klasse agiert TApplication als objektorientierter Vertreter für ein Windows-Anwendungsmodul. TApplication und TModule liefern das für eine Windows-Anwendung erforderliche grundlegende Verhalten. Elementfunktionen von TApplication erzeugen Instanzen einer Klasse, erzeugen Hauptfenster und verarbeiten Botschaften.

Öffentliche Datenelemente

HAccTable

```
HACCEL HAccTable;
```

Um Kompatibilität zu vorherigen Versionen zu gewährleisten, hält HAccTable ein Handle auf die aktuelle, von der Anwendung verwendete

Windows-Tastenkürzeltabelle fest. Neue Anwendungen sollten besser das Tastenkürzeltabellen-Handle TWindow verwenden.

Siehe auch: TWindow::LoadAcceleratorTable, TWindowAttr

hPrevInstanz

```
HINSTANCE hPrevInstanz;
```

Enthält das Handle der vorher ausgeführten Instanz der Windows-Anwendung. Wenn hPrevInstanz Null ist, gab es keine vorher ausgeführte Instanz, als diese Instanz die Ausführung begann. Unter Win32 ist dieser Wert immer Null.

nCmdShow

```
int nCmdShow;
```

Gibt an, wie das Hauptfenster angezeigt werden soll (entweder als Vollbild oder als Symbol). Dies korrespondiert zum Parameter CmdShow von WinMain. nCmdShow kann eine der folgenden Windows-API-Konstanten enthalten:

Konstante	Bedeutung
SW_SHOWDEFAULT	Zeigt das Standard-SW_xxxx-Kommando.
SW_HIDE	Versteckt das Fenster.
SW_MINIMIZE	Stellt das angegebene Fenster als Symbol dar.
SW_SHOW	Aktiviert ein Fenster mit Hilfe der aktuellen Größe und Position.
SW_SHOWMAXIMIZED	Zeigt das Fenster als Vollbild an.
SW_SHOWMINIMIZED	Zeigt das Fenster als Symbol an.
SW_SHOWNA	Zeigt das Fenster in seinem aktuellen Zustand an.
SW_SHOWNOACTIVATE	Zeigt das Fenster als Symbol an.
SW_SHOWNORMAL	Zeigt das Fenster in seiner originalen Größe und Position an.
SW_SHOWSMOOTH	Zeigt das Fenster, indem es in einer Bitmap aktualisiert wird und die Bits dann auf den Bildschirm kopiert werden.

Siehe auch: ::ShowWindow

Typdefinitionen

xs

```
enum {xsUnknown, xsBadCast, xs BadTypeid; xsAlloc; xsOwl};
```

Diese Bitflags definieren die Exception-Typen, die aufgefangen bzw. unterdrückt werden. TApplication::SuspendThrow und TApplication::QueryThrow geben die Werte dieser Bitflags zurück.

Die folgende Tabelle stellt die xs-Eception- Aufzählungskonstanten dar:

Konstante	Bedeutung
xsUnknown	Unbekannte Exception
xsBadCast	Bad_cast Exception
xsBadTypeid	Bad_typeid Exception
xsMsg	Jede aus xmsg abgeleitete Exception

xsAlloc	xalloc Exception
xsOwl	TXOwl Exception

Siehe auch: TXOwl, TApplication::QueryThrow, TApplication::SuspendThrow

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TApplication(const char far* name = 0, TModule*& gModule = ::Module,
TAppDictionary* appDict = 0);
```

Dieser Konstruktor von TApplication erzeugt ein Objekt TApplication mit dem Namen name. In gModule kann der globale Modulzeiger festgelegt werden, der auf diese Anwendung zeigt. Der Parameter appDict legt fest, in welches Dictionary sich diese Anwendung einfügt. Das Standardobjekt TAppDictionary von ObjectWindows kann durch übergeben eines Zeigers auf ein benutzerdefiniertes Objekt appDict überschrieben werden.

Konstruktor

```
TApplication(const char far* name, HINSTANCE Instance,
             HINSTANCE hPrevInstance, const char far* cmdLine,
             int cmdShow, TModule*& gModule = ::Module, TAppDictionary*
appDict = 0);
```

Dieser Konstruktor von TApplication erzeugt ein TApplication-Objekt mit dem Anwendungsnamen (name), dem Instanzen-Handle der Anwendung (Instance), dem Instanzen-Handle der vorigen Anwendung (prevInstance), der aufgerufenen Kommandozeile (cmdLine) und der Zahl von Hauptfenstern, die gezeigt werden sollen (cmdShow). Der Parameter appDict legt fest, in welches Dictionary sich diese Anwendung einfügt. Das Standardobjekt TAppDictionary von ObjectWindows kann durch übergeben eines Zeigers auf ein benutzerdefiniertes Objekt appDict überschrieben werden.

Wenn Sie Ihr eigenes WinMain erzeugen wollen, dann verwenden Sie diesen Konstruktor, da er den Zugriff auf die verschiedenen von WinMain geforderten Argumente liefert. In gModule kann der globale Modulzeiger festgelegt werden, der auf diese Anwendung zeigt.

Destruktor

```
~TApplication();
```

~TApplication entfernt das TApplication-Objekt.

Siehe auch: TApplication::nCmdShow, TModule, TAppdictionary

Öffentliche Elementfunktionen

BeginModal

```
int BeginModal(TWindow* window, int flags = MB_APPIMODAL);
```

BeginModal wird zu Beginn einer modalen Botschaftsschleife eines modalen Fensters aufgerufen. Nach der Festlegung, welches Fenster zu deaktivieren ist, speichert BeginModal den aktuellen Zustand des Fensters, deaktiviert das Fenster, ruft MessageLoop auf und reaktiviert das Fenster wieder, wenn die Botschaftsschleife beendet ist. Die Flags (die dieselben sind wie die an die Windows-API-Funktion MessageBox übergebenen) legen fest, wie BeginModal das Fenster zu behandeln hat. flags kann einen der folgenden Werte annehmen:

Konstante	Bedeutung
MB_APPLMODAL	Das zu deaktivierende Fenster (das normalerweise ein Vorfahre des modalen Fensters ist) wird durch window identifiziert. Wenn window Null ist, wird kein Fenster deaktiviert.
MB_SYSTEMMODAL	Das Fenster, das systemmodal werden soll, wird durch window identifiziert.
MB_TASKMODAL	Alle Fenster auf höchster Ebene werden deaktiviert und window wird ignoriert.BeginModal gibt -1 zurück, wenn ein Fehler auftritt.

Siehe auch: TWindow

BWCCEnabled

```
boolBWCCEnabled();
```

Zeigt an, ob die BWCC (Borland Custom Controls library, Bibliothek für Dialogelemente im Borland-Stil) aktiviert ist. Gibt true zurück, wenn BWCC aktiviert ist, und false, wenn BWCC deaktiviert ist.

CanClose

```
virtual bool CanClose();
```

Gibt true zurück, wenn das Schließen der Anwendung erlaubt ist. Stan-dardmäßig ruft CanClose die Elementfunktion CanClose ihres Hauptfensters auf und gibt true zurück, wenn sowohl das Hauptfenster wie auch die Dokumentverwaltung (TDocManager) geschlossen werden können. Wenn irgendeine der CanClose-Funktionen false zurückgeben, wird die Anwen- dung nicht geschlossen. Diese Elementfunktion wird selten redefiniert. Das Verhalten zum Schließen wird normalerweise bei Bedarf in der Elementfunktion CanClose des Hauptfensters redefiniert.

Siehe auch: TWindow::CanClose, TDocManager

Condemn

```
void Condemn(TWindow* win);
```

Räumt das Fenster auf.

Ctl3dEnabled

```
bool Ctl3dEnabled() const;
```

Gibt true zurück, wenn dreidimensionale Unterstützung (Microsoft Bibliotheks-DLL für 3-D-Dialogelemente) aktiviert ist.

Siehe auch: TApplication::EnableCtl3d, TApplication::EnableCtl3dAutosubclass

EnableBWCC

```
void EnableBWCC(bool enable = true, uint Language = 0);
```

Lädt und registriert BWCC.DLL, wenn Sie 16-Bit-Anwendungen laufen lassen, oder BWCC32.DLL, wenn Sie 32-Bit-Anwendungen laufen lassen. Standardmäßig ist BWCC aktiviert. Um BWCC zu deaktivieren, müssen Sie enable auf false setzen.

Siehe auch: TDialog

EnableCtl3d

```
void EnableCtl3d(bool enable = true);
```

Aktiviert oder deaktiviert die Verwendung der CTL3D.DLL. Wenn enable true ist, lädt und registriert EnableCtl3d CTL3D.DLL, sofern sie nicht schon aktiviert ist.

Siehe auch: TApplication::Ctl3dEnabled, TApplication::EnableCtl3dAutosubclass

EnableCtl3dAutosubclass

```
void EnableCtl3dAutosubclass(bool enable);
```

Aktiviert oder deaktiviert die Verwendung der automatischen Subklasseneinordnung von CTL3D, sofern CTL3D bereits über Ctl3dEnabled aktiviert ist. Wenn automatische Subklasseneinordnung aktiviert ist, haben alle Nicht-ObjectWindows-Dialogfenster einen 3-D-Effekt. Die allgemeinen Dialogklassen und TDocManager verwenden diese Funktion, um die automatische Subklasseneinordnung vor dem Erzeugen eines Nicht-ObjectWindows-Dialogfensters einzuschalten und es damit dreidimensional erscheinen zu lassen, und um die automatische Subklasseneinordnung direkt nach dem Entfernen des Dialogfensters auszuschalten.

Siehe auch: TDialog::EvCtlColor, TApplication::EnableCtl3d, TApplication::Ctl3dEnabled

EndModal

```
void EndModal(int result);
```

EndModal wird am Ende der modalen Botschaftsschleife eines modalen Fensters aufgerufen. Setzt result auf -1, wenn ein Fehler auftritt.

Find

```
bool Find(TEventInfo &, TEqualOperator = 0);
```

TApplication verfügt nicht über eine eigene Ereignistabelle und übergibt die Ereignisbehandlung daher an den DocManager. Find ruft einen installierten TDocManager zur Ereignisbehandlung auf.

Siehe auch: TEventHandler::TEventInfo

GetBWCCModule

```
TModule* GetBWCCModule() const;
```

Liefert einen Zeiger auf das aktivierte Modul BWCC zurück.

GetCtl3dModule

```
TModule* GetCtl3dModule() const;
```

Liefert einen Zeiger auf das aktivierte Modul Ctl3d zurück.

GetDocManager

```
TDocManager* GetDocManager();
```

Gibt einen Zeiger auf das Dokumentverwaltungsobjekt zurück, das die Anwendung aufgerufen hat.

Siehe auch: TApplication::SetDocManager, TDocManager

GetMainWindow

```
TFrameWindow* GetMainWindow();
```

Gibt einen Zeiger auf das Hauptfenster der Anwendung zurück.

Siehe auch: TApplication::SetMainWindow, TFrameWindow

GetWindowPtr

```
TWindow* GetWindowPtr(HWND hWnd) const;
```

Ermittelt einen Zeiger TWindow, der mit dem Handle auf ein Fenster (hWnd) verbunden ist. Ermöglicht mehreren Anwendungen die gemeinsame Benutzung eines HWND.

Siehe auch: TWindow::GetWindowPtr

GetWinMainParams

```
void GetWinMainParams();
```

GetWinMainParams initialisiert eine statische Instanz einer Anwendung. ObjectWindows-OwlMain verwendet diese Funktion, um statische Anwendungsinstanzen zu unterstützen.

Siehe auch: TApplication::SetWinMainParams

MessageLoop

```
virtual int MessageLoop();
```

Führt die Botschaftsschleife der Anwendung durch, die während der Lebenszeit der Anwendung läuft. Fragt Windows nach Botschaften; wenn eine empfangen wird, verarbeitet MessageLoop sie durch den Aufruf von ProcessAppMsg. Wenn die Abfrage ohne eine Botschaft zurückkehrt, ruft MessageLoop IdleAction auf, um Verarbeitungen während der Idle-Zeit durchzuführen. MessageLoop ruft PumpWaitingMessages auf, um wartende Botschaften zu erhalten und zuzuteilen. MessageLoop kann unterbrochen werden, wenn BreakMessageLoop durch EndModal gesetzt ist.

Siehe auch: TApplication::BreakMessageLoop, TApplication::IdleAction, TApplication::ProcessAppMsg, TApplication::IdleAction

PostDispatchAction

```
void PostDispatchAction();
```

Wenn die Botschaftsschleife von TApplication nicht benutzt wird, sollte diese Funktion nach der Zuteilung der einzelnen Botschaften aufgerufen werden.

PreProcessMenu

```
virtual void PreProcessMenu(HMENU hmenu);
```

Ihre Anwendung kann PreProcessMenu aufrufen, um das Menü des Hauptfensters zu verarbeiten, bevor es angezeigt wurde.

Siehe auch: TDocmanager::EvPreProcessMenu, TMenu::TMenu

ProcessAppMsg

```
virtual bool ProcessAppMsg(MSG& msg);
```

Überprüft, ob in irgendeiner Form gesonderte Verarbeitung für nicht-modale Dialogfenster, Tastenkürzel und Tastenkürzel in MDI-Anwendungen erforderlich ist. Ruft die virtuelle Funktion TWindow::PreProcessMsg des Fensters auf, das die Botschaft empfängt. Wenn Ihre Anwendung keine nicht-modalen Dialogfenster erzeugt, nicht von Tastenkürzeln Gebrauch macht und keine Multidokument (MDI)-Anwendung ist, können Sie die Ausführungsgeschwindigkeit erhöhen, indem Sie diese Funktion so überschreiben, daß sie false zurückliefert.

Siehe auch: TWindow::PreProcessMsg, Struktur MSG

PumpWaitingMessages

```
bool PumpWaitingMessages();
```

Von MessageLoop aufgerufen, verarbeitet und verteilt PumpWaitingMessages alle wartenden Botschaften, bis die Schlange leer ist. Sie setzt ferner BreakMessageLoop, wenn eine WM_QUIT-Botschaft empfangen wurde.

Siehe auch: TApplication::MessageLoop, TApplication::BreakMessageLoop

QueryThrow

```
int QueryThrow();
```

QueryThrow prüft nach, ob eine Exception noch aussteht und gibt eines oder mehrere Bit-Flags in der Aufzählung xs Exception-Status zurück.

Siehe auch: Aufzählung xs Exception-Status

ResumeThrow

```
void ResumeThrow(int);
```

ResumeThrow überprüft ausstehende Exceptions und setzt sie gegebenenfalls neu ab. Rufen Sie diese Funktion jedesmal auf, wenn Sie wieder in ObjectWindows-Code von Exception-unsicherem Code aus eintreten, wo eine Exception ausgelöst worden sein könnte.

Run

```
virtual int Run();
```

Initialisiert die Instanz, indem sie InitApplication für die erste ausführende Instanz und InitInstance für alle Instanzen aufruft. Wenn die Initialisierung erfolgreich ist, ruft Run die Funktion MessageLoop auf und startet die Anwendung. Wenn außerhalb der Botschaftsschleife Exceptions ausgelöst werden, fängt Run diese Exceptions ab.

Wenn ein Fehler beim Erzeugen eines Fensters auftritt, löst Run eine TXWindow-Exception aus. Wenn Status ein Wert ungleich Null zugewiesen wurde (den ObjectWindows zur Identifizierung eines Fehlers benutzt), wird eine TXCompatibility-Exception ausgelöst.

Siehe auch: TApplication::InitApplication, TApplication::InitInstance, TApplication::MessageLoop, TXWindow, TXMessageLoop

SetWinMainParams

```
static void SetWinMainParams(HINSTANCE hInstance, HINSTANCE,
hPrevInstance, const char far* cmdLine, int cmdShow);
```

Die Standardfunktion WinMain von ObjectWindows ruft SetMainWinParams auf, damit TApplication die Parameter für die spätere Verwendung speichern kann. WinMain erzeugt eine Instanz einer Anwendung durch Aufrufen der Funktion OwlMain aus dem Benutzercode. Die Anwendungsinstanz kann dann die vorher in SetMainWinParams gesetzten Parameter übernehmen.

Siehe auch: TApplication::GetWinMainParams

SuspendThrow

```
void SuspendThrow(xalloc& x);
```

Diese Version von SuspendThrow speichert xalloc-Exception-Information.

SuspendThrow

```
void SuspendThrow(xmsg& x);
```

Diese Version von SuspendThrow speichert xmsg-Exception-Information.

SuspendThrow

```
void SuspendThrow(TXOwl& x);
```

Diese Version von SuspendThrow speichert eine Kopie einer TXOwl-Exception.

SuspendThrow

```
void SuspendThrow(int);
```

Diese Version von SuspendThrow setzt die xs-exception status Bit-Fags entsprechend der angegebenen Exception, zum Beispiel Bad_cast oder Bad_typeid.

Siehe auch: Aufzählung xs exception-Status, TXOwl

Uncondemn

```
void Uncondemn (TWindow* win);
```

Entfernt untergeordnete Fenster, die verworfen wurden, aus der Liste der verworfenen Fenster.

Siehe auch: TWindow

Geschützte Datenelemente

BreakMessageLoop

```
bool BreakMessageLoop;
```

Verursacht bei der aktuellen modalen Botschaftsschleife Abbruch und Terminierung. Wenn die aktuelle modale Botschaftsschleife die Hauptanwendung darstellt und Ihr Programm BreakMessageLoop setzt, endet Ihre Hauptanwendung.

Siehe auch: TApplication::EndModal, TApplication::MessageLoop, TApplication::PumpWaitingMessages

MessageLoopResult

```
int MessageLoopResult;
```

MessageLoopResult wird durch einen Aufruf von EndModal gesetzt. Es enthält den von MessageLoop und BeginModal zurückgegebenen Wert.

Siehe auch: TApplication::BeginModal, TApplication::EndModal, TApplication::MessageLoop

Geschützte Elementfunktionen

IdleAction

```
virtual bool IdleAction(long idleCount);
```

ObjectWindows ruft IdleAction auf, wenn keine Botschaften auf Verarbeitung warten. Sie können IdleAction für Hintergrundverarbeitung überschreiben. Die Standardaktion besteht darin, dem Hauptfenster eine Chance für Idle-Verarbeitung zu geben, solange IdleAction true zurückgibt. idleCount gibt an, wie oft IdleAction zwischen Botschaften aufgerufen wurde.

Siehe auch: TFrameWindow::IdleAction

InitApplication

```
virtual void InitApplication();
```

ObjectWindows ruft InitApplication auf, um die erste Instanz der Anwendung zu initialisieren. Für die folgenden Instanzen wird diese Elementfunktion nicht mehr aufgerufen.

Das folgende Beispielprogramm ruft InitApplication für die erste Instanz des Programms auf.

```
class TTestApp : public TApplication {
  public:
    TTestApp(): TApplication("Instance Tester")
                {strcpy(WindowTitle, "Additional  Instance");}
  protected:
    char WindowTitle[20];

    void InitApplication() {strcpy(WindowTitle, »First Instance«);}
    void InitMainWindow() {MainWindow = new TFrameWindow(0,
WindowTitle);}
};

static TTestApp App;
```

InitInstance

```
virtual void InitInstance();
```

Führt alle für die Anwendung notwendigen Instanzinitialisierungen durch. Anders als bei InitApplication, die nur für die erste Instanz einer Anwendung aufgerufen wird, wird InitInstance unabhängig davon aufgerufen, ob noch andere ausführende Instanzen der Anwendung existieren. InitInstance ruft InitMainWindow auf und erzeugt und zeigt dann das Hauptfensterelement durch TWindow::Create und TWindow::Show. Wenn das Hauptfenster nicht erzeugt werden kann, wird eine TXInvalidMainWindow-Exception ausgelöst.

Wenn Sie diese Elementfunktion redefinieren, müssen Sie sicherstellen, daß TApplication::InitInstance explizit aufgerufen wird.

Siehe auch: TApplication::InitApplication, TApplication::InitMainWindow, TApplication::Run, TModule::MakeWindow, TWindow::Show

InitMainWindow

```
virtual void InitMainWindow();
```

Standardmäßig konstruiert InitMainWindow ein generisches TFrameWindow-Objekt mit dem Namen der Anwendung als Titel. Sie können InitMainWindow redefinieren, um ein sinnvolles Hauptfensterobjekt aus TFrameWindow (oder aus einer von TFrameWindow abgeleiteten Klasse) zu konstruieren und es in MainWindow abzuspeichern. Das Hauptfenster muß ein Fenster auf höchster Ebene sein. Das heißt, es muß von TFrameWindow abgeleitet sein. Eine typische Verwendung ist:

```
virtual void TMyApp::InitMainWindow(){
   SetMainWindow = (TMyWindow(NULL, Caption));
}
```

InitMainWindow kann überschrieben werden, um eine sinnvollere Anwendung zu konstruieren.

SetDocManager

```
TFrameWindow* SetDocManager(TDocManager* docManager);
```

Setzt einen Zeiger auf das Dokumentverwaltungsobjekt, das die Anwendung aufgerufen hat.

Siehe auch: TApplication::GetDocManager, TDocManager, TFrameWindow

SetMainWindow

```
TFrameWindow* SetMainWindow(TFrameWindow* window);
```

Erzeugt ein neues Hauptfenster und setzt das Flag WM_MAINWINDOW des Fensters.

Siehe auch: TApplication::GetMainWindow, TFrameWindow

TermInstance

```
virtual int TermInstance(int status);
```

Behandelt die Terminierung jeder ausführenden Instanz einer ObjectWindows-Anwendung.

Klasse TApplication::TXInvalidMainWindow applicat.h

Eine eingebettete Klasse mit Namen TXInvalidMainWindow beschreibt eine Exception, die von einem ungültigen Fenster herrührt. Diese Exception wird ausgelöst, wenn nicht genügend Speicher zum Erzeugen eines Fensters oder eines Dialogobjektes zur Verfügung steht. InitInstance löst

diese Exception aus, wenn sie die Instanz eines Anwendungsobjektes nicht initialisieren kann.

Öffentliche Konstruktoren

Konstruktor

```
TXInvalidMainWindow();
```

Konstruiert ein TXInvalidMainWindow-Objekt mit einer Standardbotschaft IDS_INVALIDMAINWINDOW.

Öffentliche Elementfunktionen

Clone

```
virtual TXOwl* Clone();
```

Legt eine Kopie eines Exception-Objektes an. Clone muß in jeder von TXOwl abgeleiteten Klasse implementiert sein.

Throw

```
virtual void Throw();
```

Löst ein Exception-Objekt aus. Throw muß in jeder von TXOwl abgeleiteten Klasse implementiert sein.

Klasse TAppDictionary appdict.h

Ein TAppDictionary verzeichnet Verbindungen zwischen einer Prozeßkennung (Prozeß-ID) und einer Anwendung. Eine Prozeß-ID identifiziert einen Prozeß, also ein in den Speicher geladenes, ausführungsbereites Programm (mit seinem Ausführungscode, den Daten und den Systemressourcen). Ein TAppDictionary stellt über die globale Funktion GetApplicationObject oder die Funktion GetApplication von TAppDictionary eine globale Nachschlaginstanz für Anwendungen zur Verfügung. In ObjectWindows existiert ein globales Standard-Dictionary, das exportiert wird und dann zum Einsatz kommt, wenn kein Anwendungs-Dictionary definiert wird.. Für .EXE-Dateien wird dieses globale Anwendungs-Dictionary automatisch verwendet.

TAppDictionary umfaßt eine Struktur TEntry, die die Prozeß-ID und die entsprechende, mit dieser Kennung verbundene Anwendung speichert. Die öffentlichen Elementfunktionen fügen dem Dictionary neue Einträge hinzu, finden oder entfernen diese.

Wenn ObjectWindows statisch eingebunden wird, muß nicht explizit ein Anwendungs-Dictionary erzeugt werden, weil in diesem Fall das globale Anwendungs-Standard-Dictionary von ObjectWindows verwendet wird. Wenn Sie hingegen eine DLL-Komponente schreiben und ObjectWindows in einer DLL verwenden, müssen Sie auch ein eigenes Dictionary erzeugen. Um die Definition eines Anwendungs-Dictionarys zu vereinfachen,

stellt ObjectWindows das Makro DEFINE_APP_DICTIONARY zur Verfügung, das das korrekte Dictionary für Ihre Anwendung automatisch erzeugt oder referenziert.

Beim Aufbau von .EXE-Dateien ist diese Klasse für die meisten Benutzer transparent. Komponenten-DLLs müssen jedoch für jede Task, die sie bedienen, eine eigene Instanz einer TApplication-Klasse erzeugen. Anwendungen dieser Art unterscheiden sich von .EXE-Anwendungen durch das Fehlen einer eigenen Botschaftenschleife. (Alle anderen Anwendungsdienste stehen jedoch zur Verfügung).

Obwohl eine Komponente aus mehreren DLLs bestehen kann, von denen jede über ihr eigenes TModule verfügt, besitzt die Komponente für jede Task nur eine TApplication. Ein TAppDictionary, das für alle Server (einschließlich DLL-Server) und Komponenten verwendet wird, erlaubt es Benutzern, eine vollständige und eigenständige Anwendung oder Komponente zu erzeugen. Durch die Verwendung von TAppDictionary können diese Komponenten Anwendungsobjekte gemeinsam benutzen.

Wird ObjectWindows statisch mit einer EXE verknüpft, wird die Klasse TAppDictionary als Rahmenobjekt auf einen einzigen Anwendungszeiger implementiert. Ebenso wird unter Win32 für alle Instanz-Daten verfahren. In diesem Fall sieht die Komponente niemals mehr als eine TApplication.

Um unter Verwendung der ObjectWindows DLL eine Komponenten-DLL aufzubauen, muß für diese DLL ein neues TAppDictionary-Objekt konstruiert werden. Die folgenden Schritte müssen von einer Anwendung ausgeführt werden, um die Komponenten-DLL mit dem TAppDictionary, der Anwendung und der Fensterklassenhierarchie zu verbinden:

Verwenden Sie das Makro DEFINE_APP_DICTIONARY, um eine Instanz von TAppDictionary zu erzeugen. Diese wird typischerweise eine statische Globale in einem der Module der Anwendung sein (im folgenden »AppDictionary« genannt). Mit dem Makro DEFINE_DICTIONARY ist es möglich, den gleichen Code für EXE- wie für DLL-Dateien zu verwenden.

```
TAppDictionary AppDictionary
```

Erzeugen Sie ein generisches TModule und weisen Sie es der Globalen ::Module zu. Dies ist die Voreinstellung, die in der Funktion LibMain von ObjectWindows zur Verfügung gestellt wird.

```
LibMain(...)
    ::Module = new TModule(0, hInstance);
```

Nachdem jede Instanz TApplication konstruiert worden ist, übergeben Sie als letztes Argument einen Zeiger auf das TAppDictionary. So ist sichergestellt, daß die Anwendung sich selbst in dieses Dictionary einset-

zen wird. Bei 16-Bit-DLLs muß außerdem das Argument gModule mit einem Platzhalterwert ausgestattet werden, weil die Konstruktion Module, als Resultat des in Schritt 2 ausgeführten Prozesses, zu diesem Zeitpunkt bereits abgeschlossen ist.

```
TApplication* app = new TMyApp(..., app, AppDictionary);
```

Bei Verwendung des Dok/View-Modells muß beim Konstruieren des Objekts TDocManager der Anwendungszeiger übergeben werden.

```
SetDocManager(new TDocManager(mode, this));
```

Wenn ein Fenster konstruiert wird, das nicht über ein übergeordnetes Fenster verfügt (beispielsweise das Hauptfenster), übergeben Sie als Modul die Anwendung.

```
SetMainWindow(new TFrameWindow(0, "", false, this));
```

Siehe auch:... TApplication::GetWindowPtr, TWindow::GetWindowPtr, Makro DEFINE_APP_DICTIONARY

Typdefinitionen

TEntry

```
struct TEntry {
  unsigned Pid;
  TApplication* App;
};
```

Ein Eintrag in das Anwendungs-Dictionary, der eine Prozeß-ID (Pid) mit einer Anwendung (App) verbindet. Das Dictionary wird durch (Pid) indiziert und kann nicht mehr als einen Eintrag pro Prozeßkennung besitzen.

Siehe auch: TAppDictionary::TEntry struct, TAppDictionary::Iterate

TEntryIterator

```
typedef void(*TEntryIterator)(TEntry&);
```

Ein Zeigertyp einer Dictionary-Iteratorfunktion, der eine Referenz auf einen Eintrag empfängt. Sie können der Funktion Iterate eine Funktion dieses Typs übergeben, um durch die Einträge im Dictionary zu iterieren.

Siehe auch:...TAppDictionary::TEntryIterator typedef

Öffentlicher Konstruktor und Destruktor

Konstruktor

```
TAppDictionary();
```

Erzeugt ein Objekt TAppDictionary.

Destruktor

```
~TAppDictionary();
```

Entfernt das Objekt TAppDictionary und ruft DeleteCondemned auf, um die verworfenen Anwendungen zu beseitigen.

Öffentliche Elementfunktionen

Add

```
void Add(TApplication* app, unsigned pid = 0);
```

Fügt diesem Dictionary ein Anwendungsobjekt (app) und die entsprechende Prozeßkennung hinzu. Die Standardkennung ist die ID des aktuellen Prozesses.

Siehe auch: TAppDictionary::Remove

Condemn

```
void Condemn(TApplication* app);
```

Kennzeichnet eine Anwendung in diesem Dictionary als verworfen, indem es die Prozeßkennung auf 0 setzt. Dadurch wird die Anwendung zum Löschen durch DeleteCondemned freigegeben.

Siehe auch: TAppDictionary::DeleteCondemned

DeleteCondemned

```
bool DeleteCondemned();
```

Löscht alle verworfenen Anwendungen und die mit ihnen verbundenen Prozeßkennungen aus dem Dictionary. DeleteCondemned gibt true zurück, wenn sich keine Anwendungen mehr im Verzeichnis befinden.

Siehe auch: TAppDictionary::Condemn

GetApplication

```
TApplication* GetApplication(unsigned pid = 0);
```

Schlägt die mit einer angegebenen Prozeß-ID verbundene Anwendung nach und liefert sie zurück. Die Standardkennung ist die ID des aktuellen Prozesses. Wenn keine Anwendung mit der Prozeß-ID verbunden ist, gibt GetApplication 0 zurück.

Iterate

```
void Iterate(TEntryIterator iter);
```

Iteriert durch eine Liste von Einträgen im Anwendungs-Dictionary und ruft für jeden Eintrag die Callback-Funktion iter auf.

Siehe auch: TAppDictionary::TEntryIterator

Remove

```
void Remove(TApplication* app);
```

Sucht nach einem Eintrag im Dictionary und verwendet die Anwendung (app) zur Identifizierung. Entfernt dann die Einträge der angegebenen Anwendung und ihrer Prozeßkennung aus dem Dictionary, löscht die Anwendung jedoch nicht.

Remove

```
void Remove(unsigned pid);
```

Sucht nach dem Eintrag im Dictionary und verwendet dazu die angegebene Prozeßkennung (pid). Entfernt dann die Einträge der angegebenen Anwendung und ihrer Prozeßkennung aus dem Dictionary, löscht die Anwendung jedoch nicht.

Siehe auch: TAppDictionary::Add

Klasse TAutoFactory<> olefacto.h

Das Klassen-Template TAutoFactory<> erzeugt Callback-Code für ObjectWindows-Klassen. Die Anwendungsklasse wird dem Template als Argument übergeben. TAutoFactory<> bietet von sich aus keine Unterstützung für die Verknüpfung und Einbettung von automatisierten ObjectWindows-Anwendungen.

TAutoFactory<> erleichtert das Schreiben von Callback-Funktionen. Aber natürlich können Sie eigene Callback-Funktionen schreiben oder alternative Implementierungen für jede beliebige Funktion von TAutoFactory<> umsetzen.

Siehe auch: Typdefinition TComponentFactory, Klasse TOcRegistrar, Klasse TOleFactoryBase

Öffentliche Elementfunktionen

Create

```
static IUnknown* Create(IUnknown* outer, uint32 options, uint32 id);
```

Create ist eine TComponentFactory-Callback-Funktion, die die Anwendung erzeugt oder sie entfernt oder die Objekte erzeugt. Create erzeugt ein neues Anwendungsobjekt, wenn ein solches noch nicht existiert. Das Argument outer zeigt auf die OLE 2-Schnittstelle IUnknown, mit der dieses Objekt sich aggregiert. Wenn outer gleich 0 ist, wird das neue Objekt entweder nicht aggregiert, oder es wird das Hauptobjekt.

Das Argument options zeigt den Ausführungsmodus der Anwendung an. Die Werte für options werden entweder von der Kommandozeile oder von ObjectComponents gesetzt. Sie werden dieser Callback-Funktion vom »Registrar« übergeben. Die Anwendung ermittelt anhand dieser Flags, wie sie sich auszuführen hat, und die Factory Callback-Funktion erfährt aus ihnen, was sie zu tun hat. Zum Beispiel zeigt der Wert amExeMode an, daß der Server als .EXE ausgeführt wird, entweder, weil er als .EXE compiliert wurde, oder weil er eine .DLL ist, die von einem .EXE-Stub gestartet wurde und nun als ausführbares Programm läuft. Eine Beschreibung der Werte, die das Argument options annehmen kann, findet sich im Aufzählungstyp TOcAppMode.

Wenn die Anwendung bereits besteht, gibt Create ihre OLE-Schnittstelle zurück und registriert die Optionen vom Aufzählungstyp TOcAppMode, der auf OLE-bezogene Flags enthält, die in der Kommandozeile der Anwendung verwendet werden. Beispielsweise weist das Flag amAutomation den Server an, sich selbst als Einzelbenutzeranwendung zu registrieren. (Im allgemeinen teilen diese Flags einer Anwendung mit, ob sie als Server ausgeführt wird, ob sie sich selbst registrieren muß, usw.)

Das Argument id, das für TAutoFactory nicht verwendet wird, ist immer 0.

Siehe auch: TAutoFactory::DestroyApp, Aufzählungstyp TOcAppMode

CreateApp

```
static T* CreateApp(uint32 options);
```

CreateApp erzeugt eine neue automatisierte Anwendung. In Voreinstellung - ohne Argumente - erzeugt es eine neue Anwendung vom Template-Typ T. Das Argument options ist ein Wert des Aufzählungstyps TOcAppMode, wie zum Beispiel amRun, amAutomation usw., der den Ausführungsmodus der Anwendung anzeigt.

Siehe auch: TAutoFactory::DestroyApp, Aufzählungstyp TOcAppMode

DestroyApp

```
static void DestroyApp(T* App);
```

Entfernt die in app angegebene, zuvor erzeugte Anwendung.

operator TComponentFactory

```
operator TComponentFactory();
```

Wandelt das Objekt in einen Zeiger auf die Factory um. ObjectComponents verwendet diesen Zeiger, um das automatisierte Objekt zu erzeugen.

Siehe auch: TAutoFactory::CreateApp

Klasse TBandInfo dc.h

Ein ObjectWindows-struct, äquivalent zu BANDINFOSTRUCT, mit Namen TBandInfo dient der Übergabe von Information an einen Druckertreiber, der eine abschnittweise Bearbeitung unterstützt. TBandInfo ist wie folgt deklariert:

```
struct TBandInfo {

 bool HasGraphics;
  bool HasText;
  TRect GraphicsRect;
};
```

HasGraphics ist true, wenn sich Grafiken auf der Seite oder im Abschnitt befinden (oder erwartet werden). Anderenfalls ist es false. HasText ist true, wenn sich Text auf der Seite oder im Abschnitt befindet (oder erwartet wird). Anderenfalls ist es false. GraphicsRect definiert die Grenzen für alle Grafiken auf der Seite.

Siehe auch: TPrintDC::BandInfo, TPrintDC::NextBand

Klasse TBitmap gdiobjec.h

TBitmap ist die GDI-Bitmap-Klasse, die von TGdiObject abgeleitet ist. TBitMap kann eine Bitmap von vielen Quellen aus konstruieren. TBitmap-Objekte sind DDBs (device-dependent bitmaps, geräteabhängige Bitmaps), die sich von den DIBs (device-independent bitmaps, geräteunabhängige Bitmaps) unterscheiden. Letztere sind durch TDib-Objekte repräsentiert.

Öffentliche Konstruktoren

Konstruktor

```
TBitmap(HBITMAP handle, TAutoDelete autoDelete = NoAutoDelete);
```

Erzeugt ein TBitmap-Objekt und setzt das Handle-Datenelement auf das gegebene entliehene handle. Das Datenelement ShouldDelete nimmt standardmäßig false an, um sicherzustellen, daß das entliehene Handle nicht gelöscht wird, wenn das C++ Objekt entfernt wird.

Siehe auch: TGdiObject::Handle, TGdiObject::ShouldDelete

Konstruktor

```
TBitmap(const TClipboard& clipboard);
```

Erzeugt ein TBitmap-Objekt mit Werten aus der gegebenen Zwischenablage.

Siehe auch: TClipBoard::GetClipboardData

Konstruktor

```
TBitmap(const TBitmap& bitmap);
```

Erzeugt eine Kopie des gegebenen bitmap-Objektes.

Siehe auch: TBitmap::GetObject, ::CreateCompatibleBitmap, ::CreateBitmap

Konstruktor

```
TBitmap(int width, int height, BYTE planes, BYTE bitCount = 1, void far*
        bits = 0);
```

Erzeugt ein Bitmap-Objekt aus bitCount Bits im Puffer bits mit den gegebenen Argumentwerten width, height und planes.

Siehe auch: ::CreateBitmap

Konstruktor

```
TBitmap(const BITMAP far* bitmap);
```

Erzeugt ein Bitmap-Objekt mit den in der gegebenen bitmap-Struktur gefundenen Werten.

Siehe auch: ::CreateBitmapIndirect, struct BITMAP

Konstruktor

```
TBitmap(const TDC& Dc, int width, int height, bool discardable = false);
```

Erzeugt ein Bitmap-Objekt für den gegebenen Gerätekontext mit den gegebenen Argumentwerten.

Siehe auch: ::CreateDiscardableBitmap, TDC

Konstruktor

```
TBitmap(const TDC& Dc, const TDib& dib, DWORD usage = CMB_INIT);
```

Erzeugt ein Bitmap-Objekt für den gegebenen Gerätekontext mit den gegebenen Argumentwerten dib und usage.

Siehe auch: ::CreateDIBBitmap, TDC, TDib

Konstruktor

```
TBitmap(const TMetaFilePict& metaFile, TPalette& palette, const TSize&
        size);
```

Erzeugt ein Bitmap-Objekt aus dem gegebenen metaFile. Verwendet die gegebenen Argumente palette und size.

Siehe auch: TMemoryDC::PlayMetaFile, TPalette

Konstruktor

```
TBitmap(const TDib& dib, const TPalette* palette = 0);
```

Erzeugt ein Bitmap-Objekt aus den gegebenen Argumenten dib und palette. Wenn keine Palette vorliegt, wird eine von der Farbtabelle der DIB konstruierte Arbeitspalette eingesetzt. Die Standardsystempalette kann ebenso mit Hilfe von &TPalette::GetStock(TPalette::Default); übergeben werden.

Siehe auch: TScreenDC::RealizePalette, ::CreateDIBBitmap, TDib

Konstruktor

```
TBitmap(HINSTANCE Instance, TResID resID);
```

Erzeugt ein Bitmap-Objekt für die gegebene Anwendung Instance aus der gegebenen Ressource.

Siehe auch: ::LoadBitmap

Öffentliche Elementfunktionen

BitsPixel

```
uint8 BitsPixel() const;
```

Gibt die Zahl von Bits pro Pixel in dieser Bitmap zurück.

Siehe auch: TBitmap::GetObject

GetBitmapBits

```
uint32 GetBitmapBits(DWORD count, void far* bits) const;
```

Kopiert bis zu count Bits aus dieser Bitmap in den Puffer bits.

Siehe auch: ::GetBitmapBits, TBitmap::GetObject

GetBitmapDimension

```
bool GetBitmapDimension(TSize& size) const;
```

Stellt die Größe dieser Bitmap wieder her (Breite und Höhe, in Zehntelmillimetern gemessen) und setzt das Argument size. Gibt true zurück, wenn der Aufruf erfolgreich war, anderenfalls false.

Siehe auch: ::GetBitmapDimensionEx, TBitmap::SetBitmapDimension, TSize

GetObject

```
bool GetObject(BITMAP far& bitmap) const;
```

Stellt die Daten (Breite, Höhe und Farbformat) für diese Bitmap wieder her und setzt sie in der gegebenen BITMAP-Struktur. Um das Bitmuster wiederherzustellen, verwenden Sie GetBitmapBits.

Siehe auch: TBitMap::GetBitmapBits, struct BITMAP

Height

```
int Height() const;
```

Gibt die Höhe der Bitmap zurück.

Siehe auch: TBitmap::GetObject

Operator<<

```
TClipboard& operator<<(TClipboard& clipboard, TBitmap& bitmap);
```

Kopiert das gegebene bitmap in das gegebene Argument clipboard. Gibt eine Referenz auf die resultierende Zwischenablage zurück, die ein normales Anketten von **<<** erlaubt.

Operator HBITMAP()

```
operator HBITMAP() const;
```

Umwandlungsoperator. Konvertiert dieses Handle der Bitmap in den Typ HBITMAP (der Windows-Datentyp repräsentiert das Handle auf eine physikalische Bitmap).

Planes

```
uint8 Planes() const;
```

Gibt die Zahl von Ebenen in dieser Bitmap zurück.

Siehe auch: TBitmap::GetObject

SetBitmapBits

```
uint32 SetBitmapBits(DWORD count, const void far* bits);
```

Kopiert bis zu count Bits aus dem Puffer bits in diese Bitmap.

Siehe auch: ::GetBitmapBits

SetBitmapDimension

```
bool SetBitmapDimension(const TSize& size, TSize* oldSize=0);
```

Setzt die Größe dieser Bitmap vom gegebenen Argument size (Breite und Höhe, gemessen in Zehntelmillimetern). Die vorherige Größe wird im Argument oldSize gesetzt. Gibt true zurück, wenn der Aufruf erfolgreich ist, andernfalls false.

Siehe auch: ::SetBitmapDimensionEx, TBitmap::GetBitmapDimension, TSize

ToClipboard

```
void ToClipboard(TClipboard& clipboard);
```

Kopiert diese Bitmap in die gegebene Zwischenablage.

Siehe auch: TClipboard::SetClipboardData

Width

```
int Width() const;
```

Gibt die Breite dieser Bitmap zurück.

Siehe auch: TBitmap::GetObject

Geschützte Elementfunktionen

Create

```
void Create(const TDib& dib, const TPalette &palette);
void Create(const TBitmap &src);
```

Erzeugt ein Bitmap-Handle aus den gegebenen Argumentobjekten.

Klasse TBitmapGadget — bitmapga.h

Als eine von TGadget abgeleitete Klasse stellt TBitmapGadget ein einfaches Gadget dar, das jeweils ein Bitmap-Bild aus einem Array mit Bildern darstellen kann.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TBitmapGadget(TResId bmpResId, int Id, TBorderStyle borderStyle,
              TResId bitmapName, int numImages, int startImage);
```

Konstruiert ein TBitmapGadget und setzt das aktuelle Bild auf das erste Bild (startImage)im Bild-Array. Setzt anschließende den Rahmenstil auf den aktuellen TGadget-Rahmenstil sowie numImages auf die Zahl von Bildern im Array.

Destruktor

```
~TBitmapGadget();
```

Löscht das Bild-Array.

Öffentliche Elementfunktionen

SelectImage

```
int SelectImage(int imageNum, bool immediate);
```

Bestimmt das aktuelle Bild und zeichnet den Client-Bereich neu, wenn das Bild geändert wurde. Aktualisiert den Client-Bereich, wenn das Bild geändert wurde.

SysColorChange

```
void SysColorChange();
```

Wenn die Systemfarben geändert wurden, wird SysColorChange durch die Funktion EvSysColorChange des Gadgetfensters aufgerufen, so daß die Bitmap-Gadgets neu aufgebaut und gezeichnet werden können.

Geschützte Elementfunktionen

GetDesiredSize

```
void GetDesiredSize(TSize& size);
```

Ruft TGadget::GetDesiredSize auf, die bestimmt, wie groß das Bitmap-Gadget werden kann. Das Gadgetfenster sendet diese Botschaft, um die Größe des Gadget zu erfragen. Wenn Größenanpassung verlangt wird, gibt GetDesiredSize die Größe zurück, die benötigt wird, um in die Rahmen und

Grenzen zu passen. Wenn es nicht verlangt wird, gibt sie die aktuelle Breite und Höhe des Gadget zurück. TGadgetWindow benötigt diese Information, um zu bestimmen, wie groß das Gadget sein muß. Sie kann diese Dimensionen bei Bedarf anpassen. Wenn WideAsPossible true ist, dann wird der Breitenparameter (size.cx) ignoriert.

Paint

```
void Paint(TDC& dc);
```

Zeichnet den Rahmen des Gadget und den Inhalt der Bitmap.

Siehe auch: TGadget::Paint

SetBounds

```
void SetBounds(TRect& r);
```

Ruft TGadget::SetBounds auf und übergibt die Dimensionen des Bitmap-Gadget. SetBounds informiert das Dialogelement-Gadget über eine Änderung an seinem begrenzenden Rechteck.

Siehe auch: TGadget::SetBounds

Klasse TBitSet — bitset.h

TBitSet setzt oder löscht ein einzelnes Bit oder eine Gruppe von Bits. Sie können diese Klasse verwenden, um Options-Flags zu setzen und zu löschen und um Information über eine Anzahl von Bits zu erhalten. Die Klasse TCharSet führt ähnliche Operationen für eine Zeichenkette durch.

Konstruktoren

Konstruktor

```
TBitSet();
```

Konstruiert ein TBitSet-Objekt.

Konstruktor

```
TBitSet(const TBitSet& bs);
```

Konstruiert ein TBitSet-Objekt als eine Kopie eines anderen TBitSet.

Öffentliche Elementfunktionen

DisableItem

```
void DisableItem(int item);
```

Löscht ein einzelnes Bit bei item.

DisableItem

```
void DisableItem(const TBitSet& bs);
```

Löscht eine Menge von Bits, die in bs aktiviert sind.

EnableItem

```
void EnableItem(int item);
```

Setzt ein einzelnes Bit bei item.

EnableItem

```
void EnableItem(const TBitSet& bs);
```

Setzt eine Menge von Bits, die in bs aktiviert sind.

Has

```
int Has(int item);
```

Ist Nicht-Null, wenn item in der Menge von Bits vorkommt.

IsEmpty

```
int TBitSet::IsEmpty();
```

Ist Nicht-Null, wenn die Menge leer ist, anderenfalls Null.

Operator +=

```
TBitSet& operator +=(int item);
```

Ruft EnableItem auf, um ein Bit in der kopierten Menge zu setzen. Gibt eine Referenz auf das kopierte TBitSet-Objekt zurück.

Operator +=

```
TBitSet& operator +=(const TBitSet& bs);
```

Ruft EnableItem auf, um die Bits zu setzen, die in bs aktiviert sind. Gibt eine Referenz auf das kopierte TBitSet-Objekt zurück.

Operator -=

```
TBitSet& operator -=(int item);
```

Ruft DisableItem auf, um ein Bit in der Menge zu Löschen. Gibt eine Referenz auf das kopierte TBitSet-Objekt zurück.

Operator -=

```
TBitSet& operator -=(const TBitSet& bs);
```

Ruft DisableItem auf, um die Bits zu löschen, die in bs aktiviert sind. Gibt eine Referenz auf das kopierte TBitSet-Objekt zurück.

Operator &=

```
TBitSet& operator &=(const TBitSet&);
```

Verbindet alle Bits in der kopierten Menge durch »Und« und gibt eine Referenz auf das kopierte TBitSet-Objekt zurück.

Operator |=

```
TBitSet& operator |=(const TBitSet&);
```

Verbindet alle Bits in der kopierten Menge durch »Oder« und gibt eine Referenz auf das kopierte TBitSet-Objekt zurück.

Operator ~

```
TBitSet operator ~(const TBitSet&);
```

Gibt eine Menge von Bits zurück, die das Gegenteil der angegebenen Menge von Bits darstellt. Wenn die Bitmenge zum Beispiel 01010101 ist, wird die zurückgegebene Menge 10101010 sein. Gibt eine Referenz auf das kopierte TBitSet-Objekt zurück.

Klasse TBIVbxLibrary vbxctl.h

TBIVbxLibrary ist in vbxctl.h definiert und stellt eine virtuelle Ableitung aus TModule dar. Sie verwaltet das Laden und Initialisieren von BIVBX10.DLL.Wenn Sie VBX-Dialogelemente verwenden wollen, erzeugen Sie ein Objekt TBIVbxLibrary mit dem gleichen Gültigkeitsbereich wie Ihre Anwendung. Zum Beispiel

```
int OwlMain(int, char**)
{
  TBIVbxLibrary vbxLib;
  return TTestApp().Run();
}
```

Öffentlicher Konstruktor und Destruktor

Konstruktor

```
TBIVbxLibrary();
```

Erzeugt ein TBIVbxLibrary-Objekt.

Destruktor

```
~TBIVbxLibrary();
```

Entfernt ein TBIVbxLibrary-Objekt.

Klasse TBrush gdiobjec.h

Die GDI-Pinselklasse ist von TGdiObject abgeleitet. TBrush bietet Konstruktoren, die auf der Grundlage expliziter Informationen einfarbige, stilisierte oder gemusterte Pinsel erzeugen. Sie kann zudem einen Pinsel indirekt aus einem entliehenen Handle erzeugen.

Öffentliche Konstruktoren

Konstruktor

```
TBrush(HBRUSH handle, TAutoDelete autoDelete = NoAutoDelete);
```

Erzeugt ein TBrush-Objekt und setzt das Datenelement Handle auf das gegebene entliehene handle. Das Datenelement ShouldDelete ist standard-mäßig false. Damit wird sichergestellt, daß das entliehene Handle nicht gelöscht wird, wenn das C++ Objekt entfernt wird.

Siehe auch: TGdiObject::Handle, TGdiObject::ShouldDelete

Konstruktor

```
TBrush(TColor color);
```

Erzeugt ein einfarbiges Objekt TBrush mit der angegebenen Farbe. Um eine Pinselerzeugung zu sichern, verwendet dieser Konstruktor ein Cache, das jede Farbe erkennen kann, die mit einer vordefinierten Farbe übereinstimmt.

Siehe auch: ::CreateSolidBrush, TColor

Konstruktor

```
TBrush(TColor color, int style);
```

Erzeugt ein schraffiertes TBrush-Objekt mit gegebenem Stil und Farbe.

Siehe auch: ::CreateHatchedBrush, ::CreateHashBrush, TColor

Konstruktor

```
TBrush(const TBitmap& pattern);
```

Erzeugt ein gemustertes TBrush-Objekt mit dem gegebenen Muster.

Siehe auch: ::CreatePatternBrush

Konstruktor

```
TBrush(const TDib& pattern);
```

Erzeugt ein gemustertes Objekt TBrush mit dem gegebenen DIB-Muster.

Siehe auch: ::CreateDIBPatternBrushPt, ::CreateDIBPatternBrush

Konstruktor

```
TBrush(const LOGBRUSH far* logBrush);
```

Erzeugt ein TBrush-Objekt mit Werten aus dem gegebenen logBrush.

Siehe auch: TBitmap, TColor, TDib, TGdiObject::Handle, TGdiObject::ShouldDelete, Struktur LOGBRUSH

Öffentliche Elementfunktionen

GetObject

```
bool GetObject(LOGBRUSH far& logBrush) const;
```

Fordert Informationen über dieses Pinselobjekt an und plaziert sie in der gegebenen LOGBRUSH-Struktur. Gibt true zurück, wenn der Aufruf erfolgreich ist, anderenfalls false.

Siehe auch: TGdiObject::GetObject, Struktur LOGBRUSH

Operator HBRUSH()

```
operator HBRUSH() const;
```

Umwandlungsoperator. Konvertiert das Handle des Pinsels in den Typ HBRUSH (den Windows-Datentyp, der das Handle in einen physikalischen Pinsel repräsentiert).

UnrealizeObject

```
bool UnrealizeObject();
```

Veranlaßt das GDI, daß der Ursprung dieses Pinsels bei seiner nächsten Auswahl zurückgesetzt wird. Gibt true zurück, wenn der Aufruf erfolgreich ist, andernfalls false.

Klasse TButton button.h

TButton ist eine Oberflächenklasse, die ein Aktionsschalter-Oberflächenelement in Windows repräsentiert. Sie müssen TButton verwenden, um ein Schalterdialogelement in einem übergeordneten TWindow zu erzeugen. Sie können ebenso TButton verwenden, um die Kommunikation zwischen Ihrer Anwendung und den Schalterdialogelementen eines TDialog zu erleichtern. Diese Klasse ist streamfähig.

Es gibt zwei Typen von Aktionsschaltern: reguläre und voreingestellte. Reguläre Schalter haben einen dünnen Rand. Voreingestellte Schalter (diese repräsentieren die voreingestellte Aktion des Fensters) haben einen dicken Rand.

Öffentliche Datenelemente

IsDefPB

```
bool IsDefPB;
```

Zeigt an, ob der Schalter als voreingestellter Schalter betrachtet werden soll. Bei Schaltern, die durch übergeordnete Fenster gezeichnet werden, wird IsDefPB durch den Konstruktor TButton auf der Grundlage der Stileinstellungen BS_DEFPUSHBUTTON gesetzt.

Öffentliche Konstruktoren

Konstruktor

```
TButton(Window *parent, int Id, const char far *text, int X, int Y,
    int W, int H, bool isDefault = false, TModule* module = 0);
```

Konstruiert ein Schalterobjekt mit dem angegebenen übergeordneten Fenster (parent), der Dialogelement-ID (Id), dem verbundenen Text (text), der Position (X, Y) relativ zum Ursprung des Client-Bereichs des übergeordneten Fensters, der Breite (W) und der Höhe (H). Wenn IsDefault true ist, fügt der Konstruktor BS_DEFPUSHBUTTON zur vorgegebenen Eigenschaftsmenge für TButton hinzu (in Attr.Style). Anderenfalls fügt er BS_PUSHBUTTON hinzu.

Konstruktor

```
TButton(TWindow *parent, int resID, TModule* module = 0);
```

Konstruiert ein TButton-Objekt, um es mit einem Schalterdialogelement von TDialog zu assoziieren. Ruft DisableTransfer auf, um den Schalter von dem Transfermechanismus auszuschließen, da es dort keine zu transferierende Daten gibt.

Der Parameter resId muß zu einer Schalter-Ressource, die Sie definieren, korrespondieren.

Siehe auch: TControl::TControl

Geschützte Datenelemente

IsCurrentDefPB

```
bool IsCurrentDefPB;
```

Zeigt an, ob der aktuelle Schalter der voreingestellte Aktionsschalter ist.

Geschützte Elementfunktionen

BMSetStyle

```
LRESULT BMSetStyle(WPARAM, LPARAM);
```

Da ein Schalter nicht vom übergeordneten Fenster gezeichnet werden und gleichzeitig eigene Schaltereigenschaften besitzen kann, verfolgt BMSetStyle die Eigenschaften, wenn der Schalter vom übergeordneten Fenster gezeichnet wird und Windows die Eigenschaft auf BS_DEFPUSHBUTTON zu setzen versucht. BMSetStyle setzt IsCurrentDefPB auf true, wenn die Schaltereigenschaft BS_DEFPUSHBUTTON ist.

EvGetDlgCode

```
uint EvGetDlgCode(MSG far*);
```

Antwortet auf Botschaften von WM_GETDLGCODE, die an ein mit einem Dialogelement verbundenes Dialogfenster gesendet werden. EvGetDlgCode erlaubt es der Dialogverwaltung, eine Botschaft, die normalerweise direkt an ein Dialogelement gehen würde, abzufangen und das Dialogelement zu fragen, ob es diese Botschaft verarbeiten möchte. Falls nicht, verarbeitet die Dialogverwaltung die Botschaft. Der Parameter msg zeigt die Art der Botschaft an, die an die Dialogverwaltung gesendet wird - zum Beispiel eine Dialogelementbotschaft, ein Befehl oder eine Aktionsschalterbotschaft.

EvGetDlgCode gibt einen Code zurück, der anzeigt, wie der Aktionsschalter zu behandeln ist. Handelt es sich um den aktuell aktiven Aktionsschalter, gibt EvGetDialogCode entweder DLGC_DEFPUSHBUTTON oder DLGC_UNDEFPUSHBUTTON zurück.

Siehe auch: Dialogelementbotschafts-Konstanten DLGC_xxxx

GetClassName

```
char far* GetClassName();
```

Überschreibt die Funktion GetClassName von TWindow. Liefert den Namen der Registrierungsklasse von TButton, »BUTTON_CLASS« zurück, wenn BWCC aktiviert ist. Gibt den Namen »BUTTON« zurück, wenn BWCC nicht aktiviert ist.

SetupWindow

```
void SetupWindow();
```

Überschreibt die Funktion SetupWindow von TWindow.Wenn der Schalter der voreingestellte Aktionsschalter ist und vom übergeordneten Fenster gezeichnet wird, sendet SetupWindow eine DM_SETDEFID-Botschaft an das übergeordnete Fenster.

Antworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_WM_GETDLGCODE	EVGetDlgCode
EV_MESSAGE (BM_SETSTYLE, BMSetStyle)	BMSetStyle

Klasse TButtonGadget — buttonga.h

Als eine von TGadget abgeleitete Klasse repräsentiert TButtonGadget Schalter, die Sie an- oder ausklicken können. Sie können ferner Attribute

wie zum Beispiel Farbe, Eigenschaften oder Form (eingekerbt oder nicht) auf Ihre Schalter-Gadgets anwenden.

Im allgemeinen werden Schalter-Gadgets entweder als Kommando- oder Attributschalter klassifiziert. Attributschalter umfassen Schaltfelder (die als wechselseitig ausgeschlossen betrachtet werden) und Markierungsfelder (die nicht wechselseitig ausgeschlossen sind). Das öffentliche Datenelement TType zählt diese Schaltertypen auf.

TButtonGadget enthält mehrere Funktionen, mit denen das Aussehen eines Schalters verändert werden kann. Verwenden Sie SetAntialiasEdges, um das Anti-Aliasing ein- und auszuschalten, SetNotchCorners, um die Ecken mit oder ohne Kerben darzustellen und SetShadowStyle, um den Schattenstil des Schalters festzulegen.

TButtonGadget-Objekte antworten auf Mausereignisse in der folgenden Art: Wenn eine Maustaste gedrückt wurde, wird auch der Schalter gedrückt; wenn die Maustaste losgelassen wurde, wird auch der Schalter losgelassen. Kommandos können nur eingegeben werden, wenn die Maustaste oben ist (nicht gedrückt). Wenn die Maus gedrückt wird, fangen TButtonGadget-Objekte die Maus ab und reservieren alle Mausbotschaften für das aktuelle Fenster. Beim Loslassen der Maustaste wird diese Reservierung wieder aufgehoben. Das öffentliche Datenelement TState zählt die drei Schalterzustände auf.

Typdefinitionen

TShadowStyle

```
enum TShadowStyle;
```

Zählt Schatteneigenschaften von Schaltern auf – entweder einzeln (1) oder doppelt (2).

TState

```
enum TState;
```

TState zählt die drei Schalterpositionen, während derer der Schalter gedrückt werden kann auf: auf (0), ab (1) und ein Zwischenzustand (2). Ein Nicht-Null-Wert zeigt einen hervorgehobenen Schalter an.

TType

```
enum TType;
```

Zählt die Schaltertypen auf: Kommando, exklusiv oder nicht-exklusiv.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TButtonGadget(TResId bmpResId, int id, TType type = Command,
              bool enabled = false, TState state = Up,
              bool repeat = false);
```

Konstruiert ein TButtonGadget-Objekt mit Hilfe der angegebenen Bitmap-ID, Schalter-Gadget-ID und -Typ. enabled ist auf false gesetzt und

der Schalter ist nicht gedrückt. Der Schalter ist nicht aktiviert - das ist sein Ausgangszustand, bevor er durch Befehle aktiviert wird.

Destruktor

```
~TButtonGadget();
```

Destruiert ein TButtonGadget-Objekt.

Siehe auch: TButtonGadget::TState

Öffentliche Elementfunktionen

CommandEnable

```
void CommandEnable();
```

Aktiviert das Schalter-Gadget für das Auffangen von Botschaften. Ruft SendMessage auf, um eine WM_COMMAND_ENABLE-Botschaft an das dem Schalter-Gadget übergeordnete Fenster zu schicken, wobei eine TCommandEnable: EvCommandEnable-Botschaft für diesen Schalter übergeben wird.

GetButtonState

```
TState GetButtonState();
```

Gibt den Schalterzustand zurück. Bei Null ist der Schalter oben und bei Eins ist er unten. Bei Zwei liegt ein Zwischenzustand vor.

Siehe auch: TButtonGadget::TState

GetButtonType

```
TType GetButtonType();
```

Gibt den Schaltertyp zurück: 1 bei Kommandoschalter, 2 bei exklusivem oder 3 bei nicht-exklusivem Schalter.

SetAntialiasEdges

```
SetAntialiasEdges(bool anti = true);
```

Schaltet das Anti-aliasing für die abgeschrägten Ecken des Schalters ein oder aus. Die Voreinstellung ist »ein«.

SetButtonState

```
void SetButtonState(TState);
```

Setzt den Schalterzustand. Wenn sich der Zustand geändert hat, der Schalter exklusiv ist und im Zustand »ab« ist, überprüft sie, daß der Schalter exklusiv ist, setzt State und ruft Invalidate auf, um den geänderten Bereich des Gadget für ein die Neuzeichnung zu markieren.

Siehe auch: TButtonGadget::TState

SetNotchCorners

```
void SetNotchCorners(bool NotchCorners = true);
```

Standardmäßig implementiert SetNotchCorners eingekerbte Ecken für Schalter. Um den Rahmen des Schalters neu zu zeichnen, wenn das Fenster bereits erzeugt wurde, rufen Sie InvalidateRect mit dem Bounds-Rechteck auf.

Siehe auch: TButtonGadget::Invalidate, TGadget::InvalidateRect, TGadget::Paint

SetShadowStyle

```
void SetShadowStyle(TShadowStyle);
```

Setzt die Schaltereigenschaft auf einen Schattenstil, der standardmäßig DoubleShadow ist. Setzt den linken und oberen Rand auf 2 und den rechten und unteren Rand auf ShadowStyle + 1.

SysColorChange

```
void SysColorChange();
```

SysColorChange antwortet auf eine EvSysColorChange-Botschaft, die durch das eigene TGadgetWindow weitergereicht wurde, indem der Dither-Pinsel auf Null gesetzt wird. Wenn eine Bitmap der Benutzeroberfläche existiert, löscht SysColorchange die Bitmap und erzeugt sie von Neuem, um die neuen Schalterfarben zu erhalten.

Geschützte Datenelemente

AntialiasEdges

```
bool AntialiasEdges;
```

Ist true, wenn das Anti-Aliasing eingeschaltet ist.

BitmapOrigin

```
TPoint BitmapOrigin;
```

Zeigt auf die x- und y-Koordinaten der für diese Schalter-Gadgets verwendeten Bitmap.

CelArray

```
TCelArray* CelArray;
```

Das Array der Cels, die von diesem Schalter-Gadget verwendet werden.

NotchCorners

```
bool NotchCorners;
```

Ist mit 1 initialisiert. NotchCorners hat den Wert 1, wenn der Schalter gekerbte Ecken besitzt und andernfalls 0.

Pressed

```
bool Pressed;
```

Ist mit 1 initialisiert. Pressed hat den Wert 1, wenn der Schalter losgelassen ist und andernfalls 0.

Siehe auch: TButtonGadget::Activate, TButtonGadget::BeginPressed, TButtonGadget::CancelPressed

Repeat

```
bool Repeat;
```

Ist mit 1 initialisiert. Repeat speichert den Wiederholungszähler für Tastaturereignisse.

ResId

```
TResId ResId;
```

Enthält die Ressourcen-Id für die Bitmap dieses Schalter-Gadget.

ShadowStyle

```
TShadowStyle ShadowStyle;
```

Enthält den Schattenstil für den Schalter –1 für einzeln, 2 für doppelt.

State

```
TState State;
```

Enthält den Schalterzustand – entweder »auf«, »ab« oder einen Zwischenzustand.

Type

```
TType Type;
```

Enthält den Schaltertyp – entweder Kommando, exklusiv oder nicht-exklusiv.

Geschützte Elementfunktionen

Activate

```
virtual void Activate(TPoint& p);
```

Wird aufgerufen, wenn die Maus im Zustand »auf« ist. Activate setzt Pressed auf false, ändert den Zustand für Attributschalter und zeichnet den Schalter in seinem aktuellen Zustand. Dazu ruft sie CancelPressed auf, hinterlegt eine WM_COMMAND-Botschaft für das dem Gadgetfenster übergeordnete Fenster und sendet Menübotschaften an das dem Gadget-fenster übergeordnete Fenster.

Siehe auch: TButtonGadget::Pressed

BeginPressed

```
virtual void BeginPressed(TPoint& p);
```

Wenn der Mausschalter gedrückt wurde, setzt BeginPressed Pressed auf true, zeichnet den gedrückten Schalter und sendet Menübotschaften an das dem Gadgetfenster übergeordnete Fenster.

Siehe auch: TButtonGadget::Pressed

BuildCelArray

```
virtual void BuildCelArray();
```

Erzeugt ein Array von Cels und verwendet dazu die Ressourcen-Bitmap als Grundzeichenkörper. Jedes bestehende Array von Cels sollte gelöscht werden, wenn für es ein Ersatz erzeugt wird.

Siehe auch: TCelArray

CancelPressed

```
virtual void CancelPressed(TPoint& p);
```

Wenn der Mausschalter losgelassen wird, setzt CancelPressed Pressed auf false, zeichnet den Schalter und sendet Menübotschaften an das dem Gadgetfenster übergeordnete Fenster.

Siehe auch: TButtonGadget::Pressed

GetDesiredSize

```
void GetDesiredSize(TSize& size);
```

Speichert die Breite und die Höhe (gemessen in Pixeln) des Schalter-Gadget in size. Ruft GetDesiredSize von TGadget auf, um das Verhältnis zwischen zwei Rechtecken zu berechnen.

GetGlyphDib

```
virtual TDib* GetGlyphDib();
```

Stellt die glyphdib zur Verfügung. Sie können diese Funktion überschreiben, um eine andere dib zu erhalten oder um die Attribute der dib zu ändern, wie Farben usw.

Invalidate

```
void Invalidate();
```

Wenn ein Schalter gedrückt wurde oder sich der Schalterzustand geändert hat, macht Invalidate den geänderten Bereich des Gadget ungültig (d.h., markiert ihn für die Neuzeichnung). Invalidate macht nur den geänderten Bereich ungültig. Um das gesamte Gadget neu zu zeichnen, rufen Sie TGadget::InvalidateRect auf und übergeben die Grenzen des Rechtecks.

Siehe auch: TButtonGadget::TState, TGadget::InvalidateRect

LButtonDown

```
void LButtonDown(uint modKeys, TPoint& p);
```

Überschreibt eine TGadget-Elementfunktion und antwortet auf das Drücken des linken Mausschalters mit dem Aufruf von BeginPressed.

Siehe auch: TButtonGadget::BeginPressed

LButtonUp

```
void LButtonUp(uint modKeys, TPoint& p);
```

Überschreibt TGadget-Elementfunktionen und antwortet auf das Loslassen des linken Mausschalters mit dem Aufruf von Activate.

Siehe auch: TButtonGadget::Activate

MouseEnter

```
void MouseEnter(uint modKeys, TPoint& p);
```

Wird aufgerufen, wenn der Mauszeiger in den Bereich des Schalter-Gadgets eintritt. modKeys zeigt die virtuelle Tasteninformation an und kann eine beliebige Kombination der folgenden Werte enthalten: MK_CONTROL, MK_LBUTTON, MK_MBUTTON, MK_RBUTTON oder MK_SHIFT. p zeigt an, wo die Maus in den Bereich des Schalter-Gadgets eingetreten ist.

MouseLeave

```
void MouseLeave(uint modKeys, TPoint& p);
```

Wird aufgerufen, wenn der Mauszeiger den Bereich des Schalter-Gadgets verläßt. modKeys zeigt die virtuelle Tasteninformation an und kann eine beliebige Kombination der folgenden Werte enthalten: MK_CONTROL, MK_LBUTTON, MK_MBUTTON, MK_RBUTTON oder MK_SHIFT. p zeigt an, wo die Maus den Bereich des Schalter-Gadgets verlassen hat.

MouseMove

```
void MouseMove(uint modKeys, TPoint& p);
```

Ruft TGadget::MouseMove als Antwort auf eine gezogene Maus. Wenn die Maus den Schalter verläßt, ruft MouseMove CancelPressed auf. Wenn sich die Maus auf den Schalter zurückbewegt, ruft MouseMove BeginPressed auf.

Siehe auch: TButtonGadget::BeginPressed, TButtonGadget::CancelPressed

Paint

```
void Paint(TDC& dc);
```

Ruft ::GetSystemMetrics auf, um die Breite und Höhe des Fensterrahmens (in Pixeln) zu erhalten. Ruft GetImageSize auf, um die Größe der Bitmap zu erhalten und setzt das innere Rechteck auf die angegebenen Dimensionen. Ruft TGadget::PaintBorder auf, um die aktuelle Zeichnung des Dialogelementrahmens durchzuführen. Bevor das Dialogelement gezeichnet wird, bestimmt Paint, ob die Ecken des Dialogelementes eingekerbt sind, und ruft dann GetSysColor auf, um festzustellen, ob Hervorhebungen oder Farben zum Schattieren verwendet werden. Paint geht von einem flachen Rahmenstil aus. Anschließend zeichnet Paint die obere, linke, rechte und untere Seite des Dialogelements, paßt die Position der Bitmap an und beendet das Zeichnen des Dialogelements unter Verwendung des angegebenen Embossing, Fading und Dithering.

ReleaseGlyphDib

```
virtual void ReleaseGlyphDib(TDib* glyph);
```

Gibt die glyphdib wieder frei, in Abhängigkeit von der Art und Weise, wie GetGlyphDib sie zur Verfügung gestellt hat.

SetBounds

```
void SetBounds(TRect& r);
```

Erfragt die Größe der Bitmap, ruft TGadget::SetBounds auf, um die Grenzen des Rechtecks zu setzen, und zentriert die Bitmap innerhalb des Schalterrechtecks.

Siehe auch: TGadget::SetBounds

Klasse TButtonGadgetEnabler — buttonga.h

TButtonGadgetEnabler ist aus TCommandEnabler abgeleitet und dient der Befehlsaktivierung für Schalter-Gadgets. Die Funktionen dieser Klasse verändern den Text, den Markierungszustand und das Aussehen eines Schalter-Gadgets.

Öffentlicher Konstruktor

Konstruktor

```
TButtonGadgetEnabler(HWND hWndReceiver, TButtonGadget* g)
```

Konstruiert ein Objekt TButtonGadgetEnabler für das angegebene Gadget. hWndReceiver ist das Fenster, das die Botschaft empfängt.

Geschütztes Datenelement

gadget

```
TButtonGadget* gadget;
```

Das Schalter-Gadget, das aktiviert oder deaktiviert wird.

Öffentliche Elementfunktionen

Enable

```
void Enable(bool Enable);
```

Überschreibt TCommandEnable::Enable. Aktiviert oder deaktiviert die Tastatur, Mauseingaben und die Darstellung des entsprechenden Schalter-Gadgets.

SetCheck

```
void SetCheck(int state);
```

Überschreibt TCommandEnable::SetCheck. Ändert den Markierungszustand des entsprechenden Schalter-Gadgets.

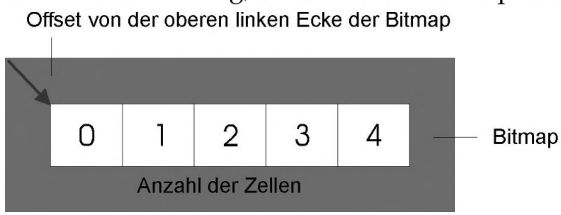
SetText

```
void SetText(const char far*);
```

Überschreibt TCommandEnable::SetText. Ändert den Text des entsprechenden Schalter-Gadgets.

Klasse TCelArray celarray.h

TCelArray ist ein horizontales Array mit Cels (einer Animationseinheit), das durch Aufschneiden eines Teils oder der gesamten Bitmap in gleichgroße Formen erzeugt wurde. Gadgets wie Schalter können ein TCelArray verwenden, um Ressourcen-Speicher zu sparen. Die Funktionen von TCelArray ermöglichen Ihnen die Kontrolle der Dimensionen jeder Cel und die Bestimmung, ob die Cel die Bitmap löschen kann.



Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TCelArray(TBitmap* bmp, int numCels, TSize celSize = 0, TPoint Offset = 0,
          TAutoDelete = AutoDelete);
```

Konstruiert ein TCelArray aus einer Bitmap, indem die Bitmap in ein horizontales Array mit Cels der angegebenen Größe aufgeschnitten wird. Wenn autoDelete true ist, kann TCelArray die Bitmap automatisch löschen. Das Datenelement ShouldDelete ist standardmäßig true. Damit ist sichergestellt, daß das Handle gelöscht wird, sobald die Bitmap entfernt wird.

Konstruktor

```
TCelArray(TDib& dib, int numCels);
```

Konstruiert ein TCelArray aus einer DIB, indem die DIB in ein horizontales Array mit gleichgroßen Cels aufgeschnitten wird.

Konstruktor

```
TCelArray(const TCelArray& src);
```

Konstruiert ein TCelArray als eine Kopie eines bereits existierenden. Wenn das originale TCelArray seine Bitmap besitzt, kopiert der Konstruktor diese Bitmap. Anderenfalls behält er eine Referenz auf die Bitmap.

Destruktor

```
virtual ~TCelArray();
```

Wenn ShouldDelete true ist (der Vorgabewert), wird die Bitmap gelöscht. Wenn ShouldDelete false ist, finden keine Aktionen statt.

Öffentliche Elementfunktionen

CelSize

```
TSize CelSize() const;
```

Gibt die Größe jeder Cel in Pixeln zurück.

CelOffset

```
TPoint CelOffset(int cels);
```

Gibt die Position der oberen linken Ecke der gegebenen Cel relativ zur oberen linken Ecke der Bitmap zurück.

CelRect

```
TRect CelRect(int cel) const;
```

Gibt die obere linke und untere rechte Ecke der gegebenen Cel relativ zur oberen linken Ecke der Bitmap zurück.

NumCels

```
int NumCels() const;
```

Gibt die Anzahl der Cels im Array zurück.

Offset

```
TPoint Offset() const;
```

Gibt den Offset des gesamten CelArray zurück.

Operator []

```
TRect operator [](int cel) const;
```

Gibt CelRect zurück.

Operator =

```
TCelArray& operator =(const TCelArray&);
```

Gibt TCelArray zurück.

Operator TBitmap&

```
operator TBitmap&();
```

Gibt eine Referenz auf die Bitmap zurück.

SetCelSize

```
void SetCelSize(TSize size);
```

Setzt die Größe jeder Cel im Array.

SetOffset

```
void SetOffset(TPoint offs);
```

Setzt den Offset für die Cels im Array.

SetNumCels

```
void SetNumCels(int numCels);
```

Setzt die Zahl von Cels im Array.

Geschützte Datenelemente

Bitmap

```
TBitmap* Bitmap;
```

Zeigt auf die Bitmap.

CSize

```
TSize CSize;
```

Die Größe einer Cel im Array.

Offs

```
TPoint Offs;
```

Enthält den Offset der oberen linken Ecke des Cel-Array von der oberen linken Ecke der Bitmap.

NCels

```
int NCels;
```

Die Anzahl der Cels im Cel-Array.

ShouldDelete

```
bool ShouldDelete;
```

Ist true, wenn der Destruktor die mit dem Cel-Array assoziierte Bitmap löschen muß.

Klasse TCharSet — bitset.h

Als eine von TBitSet abgeleitete Klasse setzt und löscht TCharSet Bytes für eine Gruppe von Zeichen. Sie können diese Klasse verwenden, um Bits in einer Gruppe von Zeichen wie zum Beispiel die Großbuchstaben von »A« bis »Z« oder die Kleinbuchstaben von »a« bis »z« zu setzen oder zu löschen. Die Klasse TBitSet führt ähnliche Operationen für eine Gruppe von Bits durch.

Öffentliche Konstruktoren

Konstruktor

```
TCharSet();
```

Konstruiert ein TCharSet-Objekt.

Konstruktor

```
TCharSet(const TCharSet&);
```

Kopierkonstruktor für ein TCharSet-Objekt.

Konstruktor

```
TCharSet(const char far* str);
```

Konstruiert ein String mit Zeichen.

Öffentliche Elementfunktionen

Operator !=

```
int operator !=(const TBitSet& bs1, const TBitSet& bs2);
```

Verbindet alle Bits im kopierten String durch »oder« und gibt eine Referenz auf das kopierte TCharSet-Objekt zurück.

Klasse TCheckBox — checkbox.h

TCheckBox ist eine streamfähige Oberflächenklasse, die ein Markierungsfeld repräsentiert. Verwenden Sie TCheckBox, um ein Markierungsfeld-Dialogelement in einem übergeordneten Fenster zu erzeugen. Sie können ferner TCheckBox-Objekte verwenden, um Markierungsfelder, die Sie in einer Dialogfenster-Ressource erzeugt haben, einfacher zu manipulieren. Markierungsfelder mit zwei Zuständen können markiert oder demarkiert werden. Markierungsfelder mit drei Zuständen besitzen einen zusätzlichen Zustand »grau«. Die Elementfunktionen von TCheckBox ermöglichen Ihnen eine einfache Kontrolle des Markierungsfeldzustandes. Ein Markierungsfeld kann in einem Gruppenfeld (ein TGroupBox-Objekt) enthalten sein, das zusammengehörige Dialogelemente gruppiert.

Öffentliche Datenelemente

Group

```
TGroupBox *Group;
```

Wenn das Markierungsfeld zu einem Gruppenfeld (d.h. einem TGroupBox-Objekt) gehört, zeigt Group auf dieses Objekt. Wenn das Markierungsfeld nicht Bestandteil einer Gruppe ist, ist Group gleich Null.

Siehe auch: TGroupBox::TGroupBox

Öffentliche Konstruktoren

Konstruktor

```
TCheckBox(TWindow *parent, int Id, const char far *title, int x,
          int y, int w, int h, TGroupBox *group = 0, TModule* module = 0);
```

Konstruiert ein Markierungsfeldobjekt mit dem angegebenen übergeordneten Fenster (parent), der Dialogelement-ID (Id), dem assoziierten Text (title), der Position relativ zum Ursprung des Client-Bereichs des übergeordneten Fensters (x, y), der Breite (w), der Höhe (h), dem assoziierten Gruppenfeld (group) und dem besitzenden Modul (module). Ruft den TButton-Konstruktor mit ähnlichen Parametern auf. Setzt Attr.Style des Markierungsfeldes auf WS_CHILD | WS_VISIBLE | WS_TABSTOP | BS_AUTOCHECKBOX.

Konstruktor

```
TCheckBox(TWindow *parent, int resourceId, TGroupBox *group = 0, TModule*
          module = 0);
```

Konstruiert ein assoziiertes TCheckBox-Objekt für das Markierungsfeld mit einer Ressourcen-ID resourceId im übergeordneten Dialogfenster. Setzt Group auf group und schaltet dann den Datentransfermechanismus durch Aufruf von EnableTransfer ein.

Siehe auch: TButton::TButton, TWindow::EnableTransfer

Öffentliche Elementfunktionen

Check

```
void Check();
```

Erzwingt die Markierung des Markierungsfelds mit Hilfe des Aufrufs von SetCheck mit dem Wert BF_CHECKED. Benachrichtigt das assoziierte Gruppenfeld (wenn es eines gibt), daß der Zustand verändert wurde.

Siehe auch: TCheckBox::GetCheck, TCheckBox::Toggle, TCheckBox::Uncheck, TGroupBox::SelectionChanged

GetCheck

```
uint GetCheck() const;
```

Gibt den Zustand des Markierungsfelds zurück.

Tabelle 2.26: TCheckBox Markierungszustände

Markierungsfeldzustand	Rückgabewert
Markiert	BF_CHECKED
Nicht markiert	BF_UNCHECKED
Grau	BF_GRAYED

Siehe auch: TCheckBox::SetCheck

GetState

```
uint GetState() const;
```

Gibt den Markierungs-, Fokus- und Hervorhebungszustand des Markierungsfelds zurück.

Siehe auch: TCheckBox::SetState

SetCheck

```
void SetCheck(uint check);
```

Erzwingt den Wechsel des Markierungsfelds in den durch Check angegebenen Zustand. Siehe Tabelle 2.26 für mögliche Werte für Check.

Siehe auch: TCheckBox::GetCheck

SetState

```
void SetState(uint state);
```

Setzt den Markierungs-, Fokus- und Hervorhebungszustand des Markierungsfelds.

Siehe auch: TCheckBox::GetState

SetStyle

```
void SetStyle(uint style, bool redraw);
```

Ändert die Eigenschaften des Markierungsfelds.

Toggle

```
void Toggle();
```

Schaltet das Markierungsfeld zwischen »markiert« und »nicht-markiert« um, sofern es sich um ein Markierungsfeld mit zwei Zuständen handelt. Schaltet das Markierungsfeld zwischen Markiert, Nicht-Markiert und Graudarstellung um, sofern es sich um ein Markierungsfeld mit drei Zuständen handelt.

Siehe auch: TCheckBox::SetCheck

Transfer

```
uint Transfer(void *buffer, TTransferDirection direction);
```

Überschreibt TWindow::Transfer. Transferiert den Zustand des Markierungsfelds zum oder vom buffer. Dabei werden die in Tabelle 2.26 angegebenen Werte verwendet. Wenn direction tdGetDate ist, wird der Zustand des Markierungsfelds zum Puffer transferiert, wenn direction tdSetData ist, ändert sich der Markierungsfeldzustand entsprechend der Einstellung im buffer.

Transfer gibt die Größe der Transferdaten in Bytes zurück. Um die Größe zu erhalten, ohne daß ein Transfer mit dem Markierungsfeld stattfindet, können Sie tdSizeData als Argument für direction verwenden.

Uncheck

```
void Uncheck();
```

Erzwingt den Zustand »nicht-markiert« für das Markierungsfeld, indem sie die Funktion SetCheck mit dem Wert BF_UNCHECKED aufruft. Benachrichtigt das assoziierte Gruppenfeld (sofern eines vorliegt), daß sich der Zustand geändert hat.

Siehe auch: TCheckBox::Check, TCheckBox::SetCheck, TCheckBox::Toggle

Geschützte Elementfunktionen

BNClicked

```
void BNClicked();
```

Antwortet auf die Benachrichtigungsbotschaft BN_CLICKED, die ein Anklicken des Markierungsfeldes durch den Benutzer anzeigt. Wenn Group nicht Null ist, ruft BNClicked die Elementfunktion SelectionChanged des Gruppenfeldes auf, um das Gruppenfeld über einen Zustandswechsel zu informieren.

Siehe auch: TGroupBox::SelectionChanged

EvGetDlgCode

```
uint EvGetDlgCode(MSG far* msg);
```

Überschreibt die Antwort von TButton auf die Botschaft WM_GETDLGCODE, eine Eingabeprozedur, die mit einem Dialogelement assoziiert ist, das kein Markierungsfeld darstellt. Dazu wird

DefaultProcessing aufgerufen. Der Parameter msg zeigt die Art der Botschaft an, die an die Dialogfensterverwaltung geschickt wird, beispielsweise ein Befehl, eine Dialogelement- oder eine Markierungsfeldbotschaft.

EvGetDlgCode gibt einen Code zurück, der anzeigt, wie das Markierungsfeld behandelt werden soll.

Siehe auch: TButton::EvGetDlgCode, TWindow::DefaultProcessing, Dialogelementbotschafts-Konstanten DLGC_xxxx.

GetClassName

```
char far* GetClassName();
```

Wenn BWCC aktiviert ist, gibt TCheckBox CHECK_CLASS zurück. Wenn BWCC nicht aktiviert ist, gibt sie »BUTTON« zurück.

Antworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_NOTIFY_AT_CHILD (BN_CLICKED, BNClicked)	BNClicked
EV_WM_GETDLGCODE	EVGetDlgCode

Klasse TChooseColorDialog chooseco.h

TChooseColorDialog-Objekte repräsentieren modale Dialogfenster als Oberflächenelemente, die die Auswahl von Farben und benutzerdefinierte Farbanpassungen ermöglichen. TChooseColorDialog kann für den Benutzer in nicht-modale Form umgewandelt werden. Dazu wird das dem Dialog übergeordnete Fenster als unsichtbares Popup-Fenster erzeugt und dieses Popup-Fenster dem Hauptfenster der Anwendung untergeordnet. TChooseColorDialog verwendet die TChooseColor::TData-Struktur, um das Dialogfenster mit der Farbauswahl des Benutzers zu initialisieren.

Öffentliche Konstruktoren

Konstruktor

```
TChooseColorDialog(TWindow* parent, TData& data, TResId templateID = 0,
                   const char far* title = 0, TModule* module = 0);
```

Konstruiert ein Dialogfenster mit dem angegebenen übergeordneten Fenster, den Daten, der Ressourcen-ID, Fenstertitel und der Bibliotheks-ID. Setzt die Attribute des Dialogfensters auf der Grundlage der Informationen in der Struktur TChooseColor::TData.

Siehe auch: TChooseColorDialog::TData

Öffentliche Elementfunktionen

SetRGBColor

```
void SetRGBColor(TColor color);
```

Setzt die aktuelle RGB-Farbe des offenen Dialogfensters durch Senden von SetRGBMsgId. Sie können SetRGBColor verwenden, um die Botschaft zu senden, daß die aktuelle Farbauswahl geändert werden soll.

Öffentliche Datenelemente

cc

```
CHOOSECOLOR cc;
```

Speichert die Länge der Struktur TChooseColorDialog, das Fenster, welches das Dialogfenster besitzt sowie den Datenblock, der das Dialog-Template beinhaltet. Es zeigt zudem auf ein Array mit 16 RGB-Werten für die benutzerdefinierten Farbfelder im Dialogfenster und legt die Initialisierungs-Flags des Dialogfensters fest.

Siehe auch: TChooseColorDialog::TData

Data

```
TData& Data;
```

Data ist eine Referenz auf das TData-Objekt, das im Konstruktor übergeben wird.

Siehe auch: TChooseColorDialog::TData

SetRGBMsgId

```
static uint SetRGBMsgId;
```

Enthält die ID der von SetRGBColor gesendeten registrierten Botschaft.

Geschützte Elementfunktionen

DialogFunction

```
bool DialogFunction(uint message, WPARAM, LPARAM);
```

Gibt true zurück, wenn eine Botschaft behandelt wird.

Siehe auch: TDialog::DialogFunction

DoExecute

```
int DoExecute();
```

Wenn kein Fehler auftritt, kopiert DoExecute Flags und Farben nach Data und gibt Null zurück. Wenn ein Fehler auftritt, gibt DoExecute IDCANCEL mit Data.Error zurück. Letzteres enthält dann den von CommDlgExtendedError zurückgegebenen Wert.

EvSetRGBColor

```
LPARAM EvSetRGBColor(WPARAM, LPARAM);
```

Antwortet auf die von SetRGBColor gesendete Botschaft, indem sie über DefaultProcessing an die originale Klasse weitergeleitet wird. Diese Ereignisbehandlung liegt nicht in der Antworttabelle vor.

Antworttabelleneinträge

Die Antworttabelle von TChooseColorDialog enthält keine Einträge.

Klasse TChooseColorDialog::TData chooseco.h

Definiert Informationen, die für die Initialisierung der Dialogfensters mit der Farbauswahl des Benutzers benötigt werden.

Öffentliche Datenelemente

Color

```
TColor Color;
```

Color gibt die Farbe an, die beim Erzeugen des Dialogfensters zunächst ausgewählt wird. Sie enthält die Farbauswahl des Benutzers, wenn das Dialogfenster geschlossen wird.

CustColors

```
TColor* CustColors;
```

CustColors zeigt auf ein Array mit 16 Farben.

Error

```
uint32 Error;
```

Wenn das Dialogfenster erfolgreich ausgeführt wurde, ist Error 0. Andernfalls enthält es einen der folgenden Codes:

Konstante	Bedeutung
CDERR_DIALOGFAILURE	Dialogfenster konnte nicht erzeugt werden.
CDERR_FINDRESFAILURE	Angegebene Ressource konnte nicht gefunden werden.
CDERR_LOADRESFAILURE	Angegebene Ressource konnte nicht geladen werden.
CDERR_LOCKRESOURCEFAILURE	Angegebene Ressource konnte nicht gesperrt werden.
CDERR_LOADSTRFAILURE	Angegebener String konnte nicht geladen werden.

Flags

```
uint32 Flags;
```

Flags kann eine Kombination der folgenden Werte sein, die die Darstellung und die Funktionsweise des Dialogfensters steuern:

Konstante	Bedeutung
CC_FULLOPEN	Das Dialogfenster erscheint vollständig, sobald es erzeugt wird.
CC_PREVENTFULLOPEN	Deaktiviert den Aktionsschalter zum Definieren eigener Farben.
CC_RGBINIT	Das Dialogfenster verwendet die in rgbResult angegebene Farbe als die anfängliche Farbauswahl.
CC_SHOWHELP	Das Dialogfenster zeigt den Aktionsschalter »Hilfe«.

Siehe auch: TChooseColorDialog::Data

Klasse TChooseFontDialog choosefo.h

Ein TChooseFontDialog repräsentiert modale Dialogfenster-Oberflächenelemente, die ein systemdefiniertes Dialogfenster erzeugen, in dem der Benutzer eine Schriftart, einen Schriftstil (wie zum Beispiel fett oder kursiv), eine Punktgröße, einen Schrifteffekt (wie zum Beispiel durchgestrichen oder unterstrichen) und eine Farbe auswählen kann. TChooseFontDialog kann für den Benutzer in nicht-modale Form umgewandelt werden. Dazu wird das dem Dialog übergeordnete Fenster als unsichtbares Popup-Fenster erzeugt und dieses Popup-Fenster dem Hauptfenster der Anwendung untergeordnet.

Öffentliche Konstruktoren

Konstruktor

```
TChooseFontDialog(TWindow* parent, TData& data,
                  TResID templateID = 0, const char far* title = 0,
                  TModule* module = 0);
```

Konstruiert ein Dialogfenster mit den angegebenen Daten, übergeordnetem Fenster, der Ressourcen-ID, dem Fenstertitel und der Bibliotheks-ID. Setzt die Attribute des Dialogfensters auf der Grundlage der Schriftart-Information in der Struktur TChooseFontDialog::TData.

Siehe auch: TChooseFontDialog::TData

Geschützte Datenelemente

cf

```
CHOOSEFONT cf;
```

Enthält Schriftattribute. cf wird mit Hilfe von Feldern in der Klasse TChooseFontDialog::TData initialisiert. Es speichert die Länge der Struktur, das Fenster, welches das Dialogfenster besitzt, und den Datenblock, der das Dialog-Template enthält. Es legt zudem die Initialisierungs-Flags des Dialogfensters fest.

Siehe auch: TChooseFontDialog::TData

Data

```
TData& Data;
```

Data ist eine Referenz auf das TData-Objekt, welches im Konstruktor übergeben wird.

Siehe auch: TChooseFontDialog::TData

Geschützte Elementfunktionen

CmFontApply

```
void CmFontApply();
```

Standardbehandlungsroutine für den dritten Aktionsschalter (den Schalter zum Ausführen) im Dialogfenster.

DialogFunction

```
bool DialogFunction(uint message, WPARAM, LPARAM);
```

Gibt true zurück, wenn eine Botschaft behandelt wird.

Siehe auch: TDialog::DialogFunction

DoExecute

```
int DoExecute();
```

Wenn kein Fehler auftritt, kopiert DoExecute die Flag-Werte und Schriftinformation nach Ddata und gibt IDOK oder IDCANCEL zurück. Wenn ein Fehler auftritt, gibt DoExecute einen Fehlercode aus dem Datenelement Error der Struktur TChooseFontDialog::TData zurück.

Siehe auch: TChooseFontDialog::TData

Antworttabelleneinträge

Die Antworttabelle TChooseFontDialog enthält keine Einträge.

Klasse TChooseFontDialog::TData — choosefo.h

Definiert Informationen, die für die Initialisierung des Dialogfensters mit der Schriftauswahl des Benutzers benötigt werden.

Öffentliche Datenelemente

Color

```
TColor Color;
```

Gibt die Schriftfarbe an, die zunächst beim Erzeugen des Dialogfensters ausgewählt wird. Enthält die Schriftfarbauswahl des Benutzers, wenn das Dialogfenster geschlossen wird.

DC

```
HDC DC;
```

Handle auf den Gerätekontext, der die Schriftarten liefert.

Error

```
uint32 Error;
```

Bei erfolgreicher Ausführung des Dialogfensters gibt Error 0 zurück. Andernfalls enthält es einen der folgenden Codes:

Konstante	Bedeutung
CDERR_DIALOGFAILURE	Dialogfenster konnte nicht erzeugt werden.
CDERR_FINDRESFAILURE	Angegebene Ressource konnte nicht gefunden werden.
CDERR_LOCKRESOURCEFAILURE	Angegebene Ressource konnte nicht gesperrt werden.
CDERR_LOADRESFAILURE	Angegebene Ressource konnte nicht geladen werden.
CDERR_LOADSTRFAILURE	Angegebener String konnte nicht geladen werden.

CFERR_MAXLESSTHANMIN	Die in SizeMax angegebene Größe ist kleiner als die in SizeMin.
CFERR_NOFONTS	Es existiert keine Schrift.

Flags

```
uint32 Flags;
```

Flags kann eine Kombination der folgenden Konstanten sein, die Darstellung und Funktionsumfang des Dialogfensters steuern:

Konstante	Bedeutung
CF_APPLY	Aktiviert Anzeige und Verwendung des Schalters zum Ausführen.
CF_ANSIONLY	Legt fest, daß die Struktur ChooseFontDialog nur die Auswahl von Schriftarten zuläßt, die den ANSI-Zeichensatz benutzen.
CF_BOTH	Veranlaßt das Dialogfenster, sowohl die verfügbaren Druckerschriftarten als auch die Bildschirmschriftarten aufzulisten.
CF_EFFECTS	Aktiviert Durchstreichen, Unterstreichen und Farbeffekte.
CF_FIXEDPITCHONLY	Erlaubt nur Äquidistanzschrift.
CF_FORCEFONTEXIST	Zeigt einen Fehler an, wenn der Benutzer eine nicht vorhandene Schrift oder Schriftart auswählt.
CF_INITTOLOGFONTSTRUCT	Verwendet die Struktur LOGFONT, auf die LogFont zum Initialisieren der Dialogfensterelemente zeigt.
CF_LIMITSIZE	Beschränkt die Schriftauswahl auf Schriften zwischen SizeMin und SizeMax.
CF_NOSIMULATIONS	Erlaubt keine GDI-Schriftsimulationen.
CF_PRINTERFONTS	Veranlaßt das Dialogfenster, nur die Schriften aufzulisten, die vom Drucker unterstützt werden, der mit dem Gerätekontext assoziiert ist.
CF_SCALABLEONLY	Erlaubt nur die Auswahl skalierbarer Schriften.
CF_SCREENFONTS	Veranlaßt das Dialogfenster, nur die vom System unterstützten Bildschirmschriften aufzulisten.
CF_SHOWHELP	Veranlaßt das Dialogfenster, den Schalter »Hilfe« anzuzeigen.
CF_TTONLY	Erlaubt nur die Auswahl von TrueType-Schriftarten.
CF_USESTYLE	Legt fest, daß Style auf einen Puffer zeigt, der die Stilattribute für die Initialisierung der Schriftstilauswahl enthält.
CF_WYSIWYG	Erlaubt nur die Auswahl von Schriften, die sowohl auf dem Drucker als auch auf dem Bildschirm verfügbar sind.

FontType

```
uint16 FontType;
```

Schrifttyp oder Name einer Schriftart.

LogFont

```
LOGFONT LogFont;
```

Attribute der Schrift.

PointSize

```
int PointSize;
```

Punktgröße der Schrift.

SizeMax

```
int SizeMax;
```

Maximale Größe der Schrift.

SizeMin

```
int SizeMin;
```

Minimale Größe der Schrift.

Siehe auch: TChooseFontDialog::Data

Style

```
char far* Style;
```

Stil der Schrift, wie zum Beispiel fett, kursiv, unterstrichen oder durchgestrichen.

Klasse TClientDC dc.h

Die von TWindowDC abgeleitete DC-Klasse TClientDC ermöglicht den Zugriff auf den Client-Bereich eines Fensters.

Öffentliche Konstruktoren

Konstruktor

```
TClientDC(HWND wnd);
```

Erzeugt ein TClientDC-Objekt mit dem gegebenen untergeordneten Fenster. Das Datenelement Wnd wird auf wnd gesetzt.

Siehe auch: TWindowDC::Wnd, TDC::TDC

Klasse TClipboard clipboar.h

TClipboard kapselt und manipuliert Zwischenablagedaten. Die Zwischenablage kann geöffnet, geschlossen oder geleert werden und zwischen ihr und dem geöffneten Fenster lassen sich Daten in verschiedenen Formaten austauschen. Ein Objekt kann in der Zwischenablage in einer Vielzahl von Formaten vorliegen, die von Bitmap bis Text reichen.

Im Normalfall ist es Aufgabe des Fensters, das Zusammenwirken mit der Zwischenablage zu organisieren. Es tut dies, indem es auf Botschaften reagiert, die zwischen dem Besitzer der Zwischenablage und der Anwendung ausgetauscht werden. Die folgenden Ereignisbehandlungsfunktionen von ObjectWindows kapseln diese Zwischenablage-Botschaften.:

EvRenderFormat - Reagiert auf eine Botschaft WM_RENDERFORMAT, die an den Besitzer der Zwischenablage gesendet wird, wenn ein bestimmtes Zwischenablageformat, das von einer Anwendung angefordert

wurde, nicht zur Verfügung gestellt worden ist. Nachdem der Besitzer der Zwischenablage die Daten im angeforderten Format geliefert hat, ruft er SetClipboardData, um sie in die Zwischenablage einzufügen.

EvRenderAllFormats - Antwortet auf eine Botschaft, die an den Besitzer der Zwischenablage gesendet wird, wenn er mit der Lieferung eines bestimmten Datenformats in Verzug geraten ist. Der Besitzer der Zwischenablage stellt daraufhin Daten in allen möglichen Formaten zur Verfügung und ruft dann SetClipboardData auf.

Das folgende Beispiel überprüft, ob sich eine Palette in der Zwischenablage befindet. Ist das der Fall, ruft TClipboard die Palette ab, führt sie aus, und schließt dann die Zwischenablage

```
    if (clipboard.IsClipboardFormatAvailable(CF_PALETTE)) {
      newPal = new TPalette(TPalette(clipboard));  // Kopie erzeugen
      UpdatePalette(true);
    }

  // Zuerst DIB-Format versuchen.
    if (clipboard.IsClipboardFormatAvailable(CF_DIB)) {
      newDib = new TDib(TDib(clipboard));        // Kopie erzeugen
      newBitmap = new TBitmap(*newDib, newPal);  // newPal==0 ist OK
    // als zweites Metadatei versuchen
    //
    } else if (clipboard.IsClipboardFormatAvailable(CF_METAFILEPICT)) {
      if (!newPal)
        newPal = new TPalette((HPALETTE)GetStockObject(DEFAULT_PALETTE));
      newBitmap = new TBitmap(TMetaFilePict(clipboard), *newPal,
                              GetClientRect().Size());

  :
  // Nimmt eine Bitmap entgegen, behält sie und arrangiert die Daten in der
  Zwischenablage.
    //
    delete Bitmap;
    Bitmap = newBitmap;

    if (!newDib)
      newDib = new TDib(*newBitmap, newPal);
  #endif
    delete Dib;
    Dib = newDib;

    delete Palette;
    Palette = newPal ? newPal : new TPalette(*newDib);
    Palette->GetObject(Colors);


    PixelWidth  = Dib->Width();

    PixelHeight = Dib->Height();
```

```
AdjustScroller();
SetCaption("(Clipboard)");

clipboard.CloseClipboard();
```

Öffentlicher Destruktor

Destruktor

```
~TClipboard();
```

Entfernt ein TClipboard-Objekt.

Öffentliche Datenelemente

DefaultProtocol

```
static const char* DefaultProtocol;
```

Zeigt auf einen String, der den Namen des vom Client benötigten Protokolls angibt. Das Standardprotokoll ist »StdFileEditing«. Dies ist der Name des Protokolls für das Linken und Einbetten von Objekten.

Siehe auch: TClipboard::QueryCreate

Öffentliche Elementfunktionen

CloseClipboard

```
void CloseClipboard()
```

Schließt die Zwischenablage, wenn IsOpen false ist. Das Schließen der Zwischenablage erlaubt anderen Anwendungen den Zugriff auf die Zwischenablage.

Siehe auch: TClipboard::CloseClipboard, ::CloseClipboard

CountClipboardFormats `int CountClipboardFormats() const;`

Gibt die Zahl der von der Zwischenablage verwendbaren Datenformattypen zurück.

Siehe auch: TClipboard::RegisterClipboardFormats

EmptyClipboard

```
bool EmptyClipboard();
```

Leert die Zwischenablage und gibt alle Handles auf die Daten der Zwischenablage frei. Gibt true zurück, wenn die Zwischenablage leer ist, oder false , wenn ein Fehler auftritt.

GetClipboard

```
static TClipboard& GetClipboard()
```

Gibt eine Referenz auf das Objekt TClipboard zurück.

GetClipboardData

```
HANDLE GetClipboardData(uint format) const;
```

Ruft die Daten aus der Zwischenablage in dem durch format angegebenen Format ab. Es werden folgende Formate unterstützt:

Wert	Bedeutung
CF_BITMAP	Daten liegen im Bitmap-Format vor.
CF_DIB	Daten sind Speicherinhalt
CF_DIF	Daten liegen in einem Datenaustauschformat vor (DIF).
CF_DSPMETAFILEPICT	Daten liegen im Metadateiformat vor.
CF_DSPTEXT	Daten liegen im Textformat vor.
CF_METAFILEPICT	Daten liegen als Metadateistruktur vor.
CF_OEMTEXT	Daten liegen als Array von Textzeichen in OEM-Zeichensatz vor.
CF_OWNERDISPLAT	Daten liegen in einem Sonderformat vor, das die Anwendung anzeigen muß.
CF_PALETTE	Daten liegen in einem Farbpalettenformat vor.
CF_PENDATA	Daten werden für ein Pen-Betriebssystem verwendet.
CF_RIFF	Daten liegen im Resource Interchange File-Format (RIFF) vor.
CF_SYLK	Daten liegen im Symbolverknüpfungsformat (SYLK) vor.
CF_TEXT	Daten sind als Textzeichen-Array gespeichert.
CF_TIFF	Daten liegen im Tag Image File-Format (TIFF) vor.
CF_WAVE	Daten liegen als Sounddatei (.WAV-Format) vor.

Siehe auch: TClipboard::SetClipboardData

GetClipboardFormatName

```
int GetClipboardFormatName(uint format, char far* formatName,
                                        int maxCount) const;
```

Fragt den Namen des durch format angegebenen, registrierten Formats ab und kopiert das Format in den Puffer, auf den formatName zeigt. maxCount gibt die maximale Länge des Formatnamens an. Wenn der Name länger als maxCount ist, wird er abgeschnitten.

Siehe auch: TClipboard::CountClipboardFormats

GetClipboardOwner

```
HWND GetClipboardOwner() const;
```

Fragt das Handle des Fensters ab, welches die Zwischenablage aktuell besitzt. gibt andernfalls 0 zurück.

GetClipboardViewer

```
HWND GetClipboardViewer() const;
```

Fragt das Handle des ersten Fensters in der View-Kette der Zwischenablage ab.

Siehe auch: TClipboard::SetClipboardViewer

GetOpenClipboardWindow

```
HWND GetOpenClipboardWindow() const;
```

Fragt das Handle des Fensters ab, welches die Zwischenablage aktuell geöffnet hat. Wenn die Zwischenablage nicht geöffnet ist, ist false der Rückgabewert. Ist die Zwischenablage einmal geöffnet, können Anwendungen ihre Daten nicht mehr verändern.

GetPriorityClipboardFormat

```
int GetPriorityClipboardFormat(uint FAR * priorityList,
                               int count) const;
```

Gibt das erste Zwischenablageformat in einer Liste zurück. priorityList zeigt auf ein Array, das eine Liste mit Zwischenablageformaten enthält. Diese Liste ist nach Priorität sortiert. Eine Beschreibung der Zwischenablageformate finden Sie unter GetClipboardFormat.

Siehe auch: TClipboard::GetClipboardData

IsClipboardFormatAvailable

```
bool IsClipboardFormatAvailable(uint format) const;
```

Zeigt an, ob das in format angegebene Format für den Benutzer in der Zwischenablage existiert. Eine Beschreibung der Zwischenablageformate finden Sie unter GetClipboardFormat.

Mit den folgenden Codezeilen läßt sich überprüfen, ob die Zwischenablage die angegebenen Formate unterstützen kann:

```
void
TBmpViewWindow::CePaste(TCommandEnabler& ce)
{
  TClipboard& clipboard = OpenClipboard();
  ce.Enable(
    clipboard && (
      clipboard.IsClipboardFormatAvailable(CF_METAFILEPICT) ||
      clipboard.IsClipboardFormatAvailable(CF_DIB) ||
      clipboard.IsClipboardFormatAvailable(CF_BITMAP)
    )
  );
  clipboard.CloseClipboard();
```

Siehe auch: TClipboard::GetClipboardData

OpenClipboard

```
HWND GetOpenClipboardWindow() const;.
```

Fragt das Handle des Fensters ab, welches die Zwischenablage aktuell geöffnet hat. Wenn die Zwischenablage nicht geöffnet ist, ist false der Rückgabewert. Ist die Zwischenablage einmal geöffnet, können Anwendungen ihre Daten nicht mehr verändern.

Siehe auch: TClipboard::CloseClipboard

Operator bool

```
operator bool() const;
```

Überprüft das Handle. Es sollte besser IsOk verwendet werden.

QueryCreate

```
bool QueryCreate(const char far* protocol = DefaultProtocol,
                 OLEOPT_RENDER renderopt = olerender_draw,
                 OLECLIPFORMAT format = 0);
```

QueryCreate bestimmt, ob das Objekt in der Zwischenablage das angegebene Protokoll und die Wiedergabeoptionen unterstützt. DefaultProtocol zeigt auf einen String mit dem vom Client benötigten Protokollnamen. renderopt gibt die Vorgaben zum Anzeigen und Drucken der Client-Anwendung für das Zwischenablageobjekt an. renderopt ist auf olerender_draw gesetzt. Damit wird angegeben, daß die Client-Anwendung die Windows-API-Funktion OleDraw aufruft, welche die Client-Bibliothek zum Erhalten und Verwalten der Datenpräsentation auffordert. format legt das von der Client-Anwendung geforderte Zwischenablageformat fest. Die Makros _OLE_H oder _INC_OLE müssen definiert sein, bevor diese Funktion benutzt werden kann.

Siehe auch: TClipboard::QueryLink

QueryLink

```
bool QueryLink(const char far* protocol = DefaultProtocol,
               OLEOPT_RENDER renderopt = olerender_draw,
               OLECLIPFORMAT format = 0);
```

QueryLink bestimmt, ob eine Client-Anwendung die Daten der Zwischenablage verwenden kann, um ein gelinktes Objekt zu produzieren, welches das angegebene Protokoll und die angegebenen Wiedergabeoptionen benutzt. Siehe TClipboard::QueryCreate für eine Beschreibung der Parameter. Die Makros _OLE_H oder _INC_OLE müssen definiert sein, bevor diese Funktion benutzt werden kann.

Siehe auch: TClipboard::QueryCreate

RegisterClipboardFormat

```
uint RegisterClipboardFormat(const char far* formatName) const;
```

Registriert ein neues Zwischenablageformat. formatName zeigt auf einen Zeichen-String, der das neue Format identifiziert. Wenn das Format registriert werden kann, zeigt der Rückgabewert das registrierte Format an. Wenn das Format nicht registriert werden kann, ist der Rückgabewert Null. Ist das Format einmal registriert, kann es als gültiges Format für den Austausch von Daten verwendet werden.

Siehe auch: TClipboard::CountClipboardFormats, TClipboard::GetClipboardFormatName

SetClipboardData

```
HANDLE SetClipboardData(uint format, HANDLE handle);
```

Setzt ein Handle auf den Datenblock an der durch handle angezeigten Position. format gibt das Format des Datenblockes an. Die Zwischenablage muß vor dem Setzen des Daten-Handles geöffnet werden. format ist ein beliebiges Format der gültigen Zwischenablageformate (beispielsweise CF_BITMAP oder CF_DIB). Eine Liste dieser Formate finden Sie unter GetClipboardData. handle ist ein Handle auf die Speicherposition, an der

die Daten gespeichert sind. Bei erfolgreicher Ausführung ist der Rückgabewert ein Handle auf die Daten. Wenn ein Fehler auftritt, ist der Rückgabewert Null. Die Zwischenablage muß erst geschlossen werden, bevor das Fenster mit den Zwischenablagedaten aktualisiert werden kann.

Siehe auch: TClipboard::GetClipboardData

SetClipboardViewer

```
HWND SetClipboardViewer(HWND Wnd) const;
```

Fügt das durch Wnd angegebene Fenster der Kette von Fenstern hinzu, die von WM_DRAWCLIPBOARD immer dann benachrichtigt werden, wenn sich der Inhalt der Zwischenablage ändert.

Siehe auch: TClipboard::GetClipboardViewer

Geschützte Datenelemente

IsOpen

```
bool IsOpen;
```

Gibt true zurück, wenn die Zwischenablage geöffnet ist..

TheClipboard

```
static TClipboard TheClipboard;
```

Enthält die aktuelle Zwischenablage.

Geschützter Konstruktor

Konstruktor

```
TClipboard();
```

Konstruiert ein TClipboard-Objekt.

Klasse TClipboardViewer — clipview.h

Registriert einen TClipboardViewer als Betrachter für die Zwischenablage, wenn das Benutzeroberflächenelement erzeugt wird, und entfernt sich selbst aus der View-Kette der Zwischenablage, wenn dieses entfernt wird.

Geschützte Datenelemente

HWndNext

```
HWND HWndNext;
```

Gibt das nächste Fenster in der Kette von Ansichten der Zwischenablage an.

Geschützte Konstruktoren

Konstruktor

```
TClipboardViewer();
```

Konstruiert ein TClipboardViewer-Objekt.

Konstruktor

```
TClipboardViewer(HWND hWnd, TModule* module = 0);
```

Konstruiert ein TClipboardViewer-Objekt mit einem Handle (hWnd) auf die Fenster, die Benachrichtigungen empfangen werden, wenn sich der Inhalt der Zwischenablage ändert.

Geschützte Elementfunktionen

DoChangeCBChain

```
TEventStatus DoChangeCBChain(HWND hWndRemoved, HWND hWndNext);
```

Überprüft, ob sich der Inhalt der Zwischenablage geändert hat, und gibt in diesem Fall die entsprechende Botschaft aus.

DoDestroy

```
TEventStatus DoDestroy();;
```

Entfernt das Fenster aus der View-Kette der Zwischenablage.

DoDrawClipboard

```
TEventStatus DoDrawClipboard();
```

Verarbeitet EvDrawClipboard-Botschaften.

EvChangeCBChain

```
void EvChangeCBChain(HWND hWndRemoved, HWND hWndNext);
```

Antwortet auf eine Botschaft WM_CHANGECBCHAIN. hWndRemoved ist ein Handle auf das Fenster, welches entfernt wurde. hWndNext ist das dem entfernten Fenster folgende Fenster.

EvDestroy

```
void EvDestroy();
```

Antwortet auf eine Botschaft WM_DESTROY, wenn ein Fenster aus der Kette von Ansichten der Zwischenablage entfernt wurde.

EvDrawClipboard

```
void EvDrawClipboard();
```

Antwortet auf eine Botschaft WM_DRAWCLIPBOARD, die an das Fenster in der Kette von Ansichten der Zwischenablage gesendet wurde, wenn sich der Inhalt der Zwischenablage ändert.

SetupWindow

```
void SetupWindow();
```

Fügt ein Fenster in die Kette von Ansichten der Zwischenablage hinzu.

Siehe auch: TWindow::SetupWindow

Antworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_WM_CHANGECBCHAIN	EvChangeCbChain
EV_WM_DESTROY	EvDestroy
EV_WM_DRAWCLIPBOARD	EvDrawClipBoard

Klasse TColor color.h

TColor ist eine Hilfsklasse, die in Verbindung mit den Klassen TPalette, TPaletteEntry, TRgbQuad und TRgbTriple verwendet wird, um alle Windows-Farboperationen zu vereinfachen. TColor besitzt zehn statische Datenelemente, die die RGB COLORREF Standardwerte repräsentieren, von Black bis White. Konstruktoren dienen dem Erzeugen von TColor-Objekten von COLORREF- und RGB-Werten, Palettenindizes, Paletteneinträgen sowie RGBQUAD- und RGBTRIPLE-Werten. Siehe die Einträge für NBits und NColors für eine Beschreibung von TColor-bezogenen Funktionen.

Öffentliche Datenelemente

Black

```
static const TColor Black;
```

Statisches TColor-Objekt mit festem, auf RGB(0, 0, 0) gesetzten Value. Siehe das RGB Windows-Makro für weitere Informationen.

Gray

```
static const TColor Gray;
```

Enthält statisches TColor-Objekt mit festem, auf RGB(128, 128, 128) gesetzten Value.

LtBlue

```
static const TColor LtBlue;
```

Enthält statisches TColor-Objekt mit festem, auf RGB(0, 0, 255) gesetzten Value.

LtCyan

```
static const TColor LtCyan;
```

Enthält statisches TColor-Objekt mit festem, auf RGB(0, 255, 255) gesetzten Value.

LtGray

```
static const TColor LtGray;
```

Enthält statisches TColor-Objekt mit festem, auf RGB(192, 192, 192) gesetzten Value.

LtGreen

```
static const TColor LtGreen;
```

Enthält statisches TColor-Objekt mit festem, auf RGB(0, 255, 0) gesetzten Value.

LtMagenta

```
static const TColor LtMagenta;
```

Enthält statisches TColor-Objekt mit festem, auf RGB(255, 0, 255) gesetzten Value.

LtRed

```
static const TColor LtRed;
```

Enthält statisches TColor-Objekt mit festem, auf RGB(255, 0, 0) gesetzten Value.

LtYellow

```
static const TColor LtYellow;
```

Enthält statisches TColor-Objekt mit festem, auf RGB(255, 255, 0) gesetzten Value.

White

```
static const TColor White;
```

Enthält statisches TColor-Objekt mit festem, auf RGB(255, 255, 255) gesetzten Value.

Siehe auch: Windows-Makro RGB

Öffentliche Konstruktoren

Konstruktor

```
TColor();
```

Der Standardkonstruktor setzt Value auf Null.

Siehe auch: TColor::Value

Konstruktor

```
TColor(COLORREF value);
```

Erzeugt ein TColor-Objekt, bei dem Value auf das gegebene value gesetzt ist.

Siehe auch: TColor::Value

Konstruktor

```
TColor(long value);
TColor(long value) : Value((COLORREF)value) {}
```

Erzeugt ein TColor-Objekt, bei dem Value auf das (COLORREF)value gesetzt ist.

Siehe auch: TColor::Value, COLORREF

Konstruktor

```
TColor(int r, int g, int b);
```

Erzeugt ein TColor-Objekt, bei dem Value auf RGB(r,g,b) gesetzt ist.

Siehe auch: TColor::Value, RGB Makro

Konstruktor

```
TColor(int r, int g, int b, int f);
```

Erzeugt ein TColor-Objekt, bei dem Value auf RGB(r,g,b) und das Flag-Byte auf f gesetzt ist.

Siehe auch: TColor::Value, PALETEENTRY Struktur

Konstruktor

```
TColor(int index);
```

Erzeugt ein TColor-Objekt, bei dem Value auf PALETTEINDEX(index) gesetzt ist.

Siehe auch: TColor::Value, PALETTEINDEX Makro

Konstruktor

```
TColor(const PALETTEENTRY far& pe);
```

Erzeugt ein TColor-Objekt, bei dem Value wie folgt gesetzt ist:

```
RGB(pe.peRed, pe.peGreen, pe.peBlue)
```

Siehe auch: TColor::Value, RGB Makro, PALETEENTRY Struktur

Konstruktor

```
TColor(const RGBQUAD far& q);
```

Erzeugt ein TColor-Objekt, bei dem Value wie folgt gesetzt ist:

```
RGB(q.rgbRed, q.rgbGreen, q.rgbBlue)
```

Siehe auch: TColor::Value, RGB Makro, RGBQUAD Struktur

Konstruktor

```
TColor(const RGBTRIPLE far& t);
```

Erzeugt ein TColor-Objekt, bei dem Value wie folgt gesetzt ist:

```
RGB(t.rgbtRed, t.rgbtGreen, t.rgbtBlue)
```

Siehe auch: TColor::Value, RGB Makro, RGBTRIPLE Struktur

Öffentliche Elementfunktionen

Blue

```
uint8 Blue() const;
```

Gibt die Blau-Komponente von Value dieser Farbe zurück.

Siehe auch: TColor::Red, TColor::Green, RGB

Flags

```
uint8 Flags() const;
```

Gibt den peFlags-Wert von Value dieses Objektes zurück.

Siehe auch: TPaletteEntry

Green

```
uint8 Green() const;
```

Gibt die Grün-Komponente von Value dieser Farbe zurück.

Siehe auch: TColor::Red, TColor::Blue, RGB

Operator==

```
bool operator==(const TColor& clrVal);
```

Gibt true zurück, wenn Value dieser Farbe mit clrValue gleich ist, andernfalls false.

Siehe auch: TColor::Value

Operator COLORREF() `operator COLORREF() const;`

Typkonvertierungsoperator, der Value zurückgibt.

Siehe auch: TColor::Value

Index

`int Index() const;`

Gibt den Indexwert zurück, der zu dem Value dieser Farbe korrespondiert. Dazu werden die oberen zwei Byte herausmaskiert. Diese Funktion wird verwendet, wenn die Farbe ein Palettenindexwert ist.

Siehe auch: TColor::Value, COLORREF

PalIndex

`TColor PalIndex() const;`

Gibt den Palettenindex zurück, der zu dem Value dieser Farbe korrespondiert. Die zurückgegebene Farbe hat das höherwertige Byte auf 1 gesetzt.

Siehe auch: TColor::Value, TColor::Index, COLORREF

PalRelative

`TColor PalRelative() const;`

Gibt den Paletten-bezogenen RGB-Wert zurück, der zu dem Value dieser Farbe korrespondiert. Die zurückgegebene Farbe hat das höherwertige Byte auf 2 gesetzt.

Siehe auch: TColor::Value, TColor::Rgb, COLORREF

Red

`uint8 Red() const;`

Gibt die Rot-Komponente von Value dieser Farbe zurück.

Siehe auch: TColor::Blue, TColor::Green, RGB

Rgb

`TColor Rgb() const;`

Gibt die explizite RGB-Farbe zurück, die zu dem Value dieser Farbe korrespondiert. Dazu wird das obere Byte herausmaskiert.

Siehe auch: TColor::Value, COLORREF

Geschützte Datenelemente

Value

`COLORREF Value;`

Der Farbwert dieses TColor-Objektes. Value kann abhängig von der Anwendung drei verschiedene Formen annehmen:

- Explizite Werte für RGB (rot, grün, blau)
- Einen Index auf eine logische Farbpalette
- Einen palettenbezogenen RGB-Wert

Klasse TComboBox combobox.h

Mit Hilfe von TComboBox können Sie ein Kombinationsfenster oder ein Kombinationsfenster-Dialogelement in einem übergeordneten TWindow erzeugen, oder Sie können die Kommunikation zwischen Ihrer Anwendung und den Dialogelementen des Kombinationsfensters von TDialog erleichtern. TComboBox-Objekte erben den größten Teil ihres Verhaltens von TListBox. Diese Klasse ist streamfähig.

Es gibt drei Typen von Kombinationsfenstern: einfach, aufklappbar und nicht editierbar. Diese Typen werden durch die Eigenschaftskonstanten CBS_SIMPLE, CBS_DROPDOWN und CBS_DROPDOWNLIST festgelegt. Wenn diese Konstanten an den Konstruktor von TComboBox übergeben werden, geben sie für Windows den Typ des zu erzeugenden Kombinationsfensterelementes an.

Öffentliche Konstruktoren

Konstruktor

```
TComboBox(TWindow *parent, int Id, int x, int y, int w, int h,
          uint32 style, uint textLen, TModule* module = 0);
```

Konstruiert ein Kombinationsfensterobjekt mit dem angegebenen übergeordneten Fenster (parent), der Dialogelement-ID (Id), der Position (x, y) relativ zum Ursprung des Client-Bereichs des übergeordneten Fensters, der Breite (w), der Höhe (h), den Eigenschaften (style) und der Textlänge (textLen).

Ruft den Konstruktor von TListBox mit ähnlichen Parametern auf und setzt dann Attr.Style wie folgt:

```
Attr.Style = WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP | CBS_SORT |
             CBS_AUTOHSCROLL | WS_VSCROLL | style;
```

Eine der folgenden Eigenschaftskonstanten für Kombinationsfenster muß in der Eigenschaftsmenge in style liegen: CBS_SIMPLE, CBS_DROPDOWN, CBS_DROPDOWNLIST, CBS_OWNERDRAWFIXED oder CBS_OWNERDRAWVARIABLE.

Konstruktor

```
TComboBox(TWindow* parent, int ResourceId, uinttextLen = 0, TModule*
module = 0);
```

Konstruiert ein Standardkombinationsfenster mit dem gegebenen übergeordneten Fenster, der Dialogelement-ID und der Textlänge.
Siehe auch: TComboBox::TextLen, TListBox::TListBox

Öffentliche Datenelemente

TextLen

```
uint TextLen;
```

Enthält die Länge des Textes in dem mit dem Kombinationsfenster assoziierten Editierfeld.

Öffentliche Elementfunktionen

AddString

```
virtual int AddString(const char far* string);
```

Fügt einem assoziierten Listenteil eines Kombinationsfensters einen String hinzu. Gibt den Index des Strings in der Liste zurück. Der erste Eintrag liegt bei Index Null. Gibt einen negativen Wert zurück, wenn ein Fehler auftritt.

Clear

```
void Clear();
```

Löscht den Text des assoziierten Editierfeldes.

ClearList

```
virtual void ClearList();
```

Löscht alle assoziierten Einträge in der assoziierten Liste aus.

DeleteString

```
virtual int DeleteString(int index);
```

Löscht den String am übergebenen Index im assoziierten Listenteil eines Kombinationsfensters. Gibt die Zahl der in der Liste verbleibenden Einträge oder einen negativen Wert zurück, wenn ein Fehler auftritt.

DirectoryList

```
virtual int DirectoryList(uint attrs, const char far* fileSpec);
```

Füllt das Kombinationsfenster mit Dateinamen aus einem angegebenen Verzeichnis.

FindString

```
virtual int FindString(const char far* find, int indexStart) const;
```

Sucht ab dem übergebenen Index nach einer Übereinstimmung. Wenn keine Übereinstimmung nach dem letzten verglichenen String gefunden wurde, wird die Suche am Anfang der Liste fortgesetzt, bis eine Übereinstimmung gefunden wurde oder bis die Liste vollständig durchsucht wurde. Gibt den Index des ersten String im assoziierten Listenteil eines Kombinationsfensters oder einen negativen Wert zurück, wenn ein Fehler auftritt.

GetCount

```
virtual int GetCount() const;
```

Gibt die Zahl von Einträgen im assoziierten Listenteil des Kombinationsfensters oder einen negativen Wert zurück, wenn ein Fehler auftritt.

GetDroppedControlRect

```
void GetDroppedControlRect(TRect& Rect) const;
```

Fordert für Kombinationsfenster die Bildschirmkoordinaten des aufklappbaren Listenfeldes an.

GetDroppedState

```
bool GetDroppedState() const;
```

Bestimmt im Fall aufklappbarer Kombinationsfenster, ob ein Listenfeld sichtbar ist.

GetEditSel

```
int GetEditSel(int &startPos, int &endPos);
```

Gibt die Start- und Endpositionen des im assoziierten Editierfeld ausgewählten Textes zurück. Gibt CB_ERR zurück, wenn das Kombinationsfenster kein Editierfeld besitzt.

GetExtendedUI

```
bool GetExtendedUI() const;
```

Bestimmt, ob das Kombinationsfenster eine erweiterte Benutzeroberfläche hat, die sich von der Standardbenutzeroberfläche auf folgende Weise unterscheidet:

- Zeigt das Listenfeld an, wenn der Benutzer das statische Textfeld anklickt.
- Zeigt das Listenfeld an, wenn der Benutzer die Taste [Unten] drückt.
- Deaktiviert das Blättern im statischen Textfeld, wenn die Elementliste nicht sichtbar ist.

Gibt true zurück, wenn das Kombinationsfenster die erweiterte Benutzeroberfläche besitzt. Andernfalls gibt sie false zurück.

Siehe auch: TComboBox::SetExtendedUI

GetItemData

```
virtual uint32 GetItemData(int index) const;
```

Gibt den 32-Bit-Wert zurück, der mit dem Element des Kombinationsfensters assoziiert ist.

Siehe auch: TListBox::GetItemData

GetItemHeight

```
int GetItemHeight(int index) const;
```

Gibt die Höhe in Pixeln der Listenelemente des Kombinationsfensters oder einen negativen Wert zurück, wenn ein Fehler auftritt.

Siehe auch: TComboBox::GetItemData, TListBox::GetItemData

GetSelIndex

```
virtual int GetSelIndex() const;
```

Gibt den Index der Listenauswahl oder einen negativen Wert zurück, wenn keiner existiert.

GetString

```
virtual int GetString(char far* str, int index) const;
```

Fragt den Inhalt des Strings an der durch index gegebenen Position ab und gibt ihn in string zurück. GetString gibt die String-Länge oder einen negativen Wert zurück, wenn ein Fehler auftritt. Der Puffer muß groß genug für den String und die terminierende Null sein.

Siehe auch: TListBox::GetString

GetStringLen

```
virtual int GetStringLen(int index) const;
```

Gibt die String-Länge (ohne die terminierende Null) des Elements an der durch index angegebenen Position zurück. Gibt einen negativen Wert zurück, wenn ein Fehler auftritt.

Siehe auch: TListBox::GetStringLen

GetText

```
int GetText(char far* str, int maxChars) const;
```

Fragt die Zahl von Zeichen im Editier- oder statischen Bereich des Kombinationsfensters ab.

GetTextLen

```
int GetTextLen() const;
```

Gibt die Textlänge (ohne die terminierende Null) des Editierfeldes oder des statischen Bereiches des Kombinationsfensters zurück.

HideList

```
void HideList();
```

Versteckt die aufklappbare Liste eines aufklappbaren oder nicht editierbaren Kombinationsfensters.

InsertString

```
virtual int InsertString(const char far* str, int index);
```

Fügt einen String in den assoziierten Listenteil eines Kombinationsfensters an der durch Index gegebenen Position ein. Gibt den Index des Strings in der Liste oder einen negativen Wert zurück, wenn ein Fehler auftritt.

Siehe auch: TListBox::InsertString

SetEditSel

```
int SetEditSel(int startPos, int endPos);
```

Wählt Zeichen zwischen startPos und endPos im Editierfeld des Kombinationsfensters aus. Gibt CB_ERR zurück, wenn das Kombinationsfenster kein Editierfeld besitzt.

SetExtendedUI

```
int SetExtendedUI(bool extended);
```

Wenn das Kombinationsfenster die erweiterte Benutzeroberfläche besitzt, wird die erweiterte Benutzeroberfläche gesetzt.

Siehe auch: TComboBox::GetExtendedUI

SetItemData

```
virtual int SetItemData(int index, uint32 data);
```

Setzt den 32-Bit-Wert, der mit dem Element von TComboBox assoziiert ist. Gibt einen negativen Wert zurück, wenn ein Fehler auftritt.

SetItemHeight

```
int SetItemHeight(int index, int height);
```

Setzt die Höhe der Listenelemente oder des Editierfeldteils in einem Kombinationsfenster. Wenn Index oder Höhe ungültig sind, wird ein negativer Wert zurückgegeben.

Siehe auch: TComboBox::GetItemHeight

SetSelIndex

```
virtual int SetSelIndex(int index);
```

Wählt einen Zeichenstring in einem Kombinationsfenster aus. index gibt den Index dieses auszuwählenden Strings an. Wenn index 0 ist, wird die erste Zeile des Listenfelds ausgewählt. Bei index -1 wird die aktuelle Auswahl entfernt. Beim Auftreten eines Fehlers wird ein negativer Wert zurückgegeben.

Siehe auch: TComboBox::GetSelIndex

SetSelString

```
virtual int SetSelString(const char far* findStr, int indexStart);
```

Wählt einen String mit Zeichen im assoziierten Listenfeld aus und setzt den Inhalt des assoziierten Editierfeldes auf den gelieferten String.

SetText

```
void SetText(const char far* string);
```

Wählt den ersten String im assoziierten Listenfeld aus, der mit dem gelieferten string beginnt. Wenn keine Übereinstimmung gefunden wird, setzt SetText den Inhalt des assoziierten Editierfeldes auf den gelieferten String und wählt ihn aus.

ShowList

```
void ShowList();
```

Zeigt die Liste eines aufklappbaren oder nicht editierbaren Kombinationsfensters.

ShowList

```
void ShowList(bool show);
```

Zeigt oder verbirgt das aufklappbare oder nicht editierbare Kombinationsfenster, je nach dem Wert von show. Zeigt die Liste an, wenn show true ist und verbirgt sie, wenn show false ist.

Siehe auch: TComboBox::HideList

Transfer

```
uint Transfer(void* buffer, TTransferDirection direction);
```

Transferiert die Elemente und die Auswahl des Kombinationsfensters vom oder zum Transferpuffer, je nachdem, ob tdSetData oder tdGetData in direction als Richtung übergeben wurde. Dabei wird davon ausgegangen, daß buffer auf eine TComboBoxData-Struktur zeigt.

Transfer gibt die Größe eines Zeigers auf eine TComboBoxData zurück. Um die Größe ohne das Transferieren von Daten abzufragen, muß Ihre Anwendung tdSizeData in direction als Richtung übergeben.

Sie müssen in Ihrem Transferpuffer einen Zeiger auf diese Strukturen verwenden. Sie können nicht Kopien der Strukturen in Ihren Transferpuffer einbetten. Ebensowenig können Sie diese Strukturen selbst als Transferpuffer verwenden.

Siehe auch: TComboBoxData, TWindow::Transfer, Window::SetupWindow

Geschützte Elementfunktionen

GetClassName

```
virtual char far* GetClassName();
```

Gibt den Namen der Windows-Registrierungsklasse von TComboBox zurück, »ComboBox«.

SetupWindow

```
void SetupWindow();
```

Richtet das Fenster ein und begrenzt den Textumfang, den der Benutzer in das Editierfeld des Kombinationsfensters eingeben kann, auf TextLen minus 1.

Klasse TComboBoxData combobox.h

Ein TComboBoxData ist ein Oberflächenobjekt, das einen Transferpuffer für ein TComboBox-Objekt repräsentiert.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TComboBoxData();
```

Konstruiert ein TComboBox-Objekt, initialisiert Strings und ItemDatas auf leere Arrays und initialisiert Selection und SelIndex zu 0.

Destruktor

```
~TComboBoxData();
```

Löscht Strings, ItemDatas und Selection .

Öffentliche Elementfunktionen

AddItemData

```
void AddItemData(DWORD itemData);
```

Fügt die benutzerdefinierten Elementdaten dem Array ItemDatas hinzu .

AddString

```
void AddString(const char *str, bool isSelected = false);
```

Fügt den angegebenen String dem Array mit Strings hinzu. Wenn IsSelected true ist, löscht AddString Selection und kopiert string in Selection hinein.

AddStringItem

```
void AddStringItem(const char* str, uint32 itemData,
                   bool isSelected = false);
```

Ruft AddItemData auf, um die Elementdaten dem Array ItemDatas zuzufügen, und ruft AddString auf, um einen String dem Array mit Strings zuzufügen.

Clear

```
void Clear();
```

Leert die Elemente Strings und ItemDatas. Setzt den Index zurück und die Werte für die ausgewählten Strings.

GetItemDatas

```
TDwordArray& GetItemDatas();
```

Liefert das Array von DWORDS zum Transferieren an ein assoziiertes Listenfeld oder von einem assoziierten Listenfeld eines Kombinationsfensters zurück.

GetSelCount

```
int GetSelCount() const;
```

Gibt die Anzahl der ausgewählten Elemente zurück, entweder 1 oder 0.

GetSelection

```
string& GetSelection();
```

Liefert den aktuell ausgewählten String (das Datenelement Selection) zum Transferieren in ein oder aus einem Kombinationsfenster.

GetSelIndex

```
int GetSelIndex();
```

Liefert den Index (das Datenelement SelIndex) des ausgewählten Elements im String-Array zurück.

GetSelString

```
void GetSelString(char far* buffer, int bufferSize) const;
```

Kopiert den ausgewählten String in einen Puffer der angegebenen Größe. bufferSize enthält die abschließende Null.

GetSelStringLength

```
int GetSelStringLength() const;
```

Gibt die Länge des aktuell ausgewählten Strings ohne die abschließende Null zurück.

GetStrings

```
TStringArray& GetStrings();
```

Liefert das Array von Strings (das Datenelement Strings) zum Transferieren in ein assoziiertes Listenfenster oder aus einem assoziierten Listenfenster eines Kombinationsfensters zurück.

ResetSelections

```
void ResetSelections();
```

Setzt den Index des ausgewählten Elements und des aktuell ausgewählten Strings zurück.

Select

```
void Select(int index);
```

Wählt das Element an der angegebenen Indexposition aus.

SelectString

```
void SelectString(const char far* str);
```

Wählt den Auswahlstring (str) aus und markiert das entsprechende Element String (wenn ein solches existiert) als ausgewählt.

Geschützte Datenelemente

ItemDatas

```
TDwordArray* ItemDatas;
```

Array mit DWORDs für den Transfer zum und vom Listenfeld, das mit dem Kombinationsfenster assoziiert ist.

Siehe auch: TComboBoxData::SetItemData, TComboBoxData::GetItemData

Selection

```
string Selection;
```

Zeigt auf den aktuell ausgewählten String, um ihn zum oder vom Kombinationsfenster zu transferieren.

SelIndex

```
int SelIndex;
```

Index des ausgewählten Elements im String-Array.

Strings

```
TStringArray Strings;
```

Array der Klasse String für den Transfer in oder aus dem assoziierten Listenfeld des Kombinationsfensters heraus .

Klasse TCommandEnabler window.h

TCommandEnabler ist eine abstrakte Basisklasse, die für die automatische Aktivierung und Deaktivierung von Befehlen verwendet wird. Aus TCommandEnabler können andere Klassen abgeleitet werden, von denen dann jede über ihren eigenen Befehlsaktivierer verfügt. TButtonGadgetEnabler ist beispielsweise eine Ableitungsklasse, die als Befehlsaktivierer für Schalter-Gadgets dient, und TMenuItemEnabler - auch eine abgeleitete Klasse - fungiert als Befehlsaktivierer für Menüelemente. Obwohl Ihre abgeleiteten Klassen wahrscheinlich nur die Funktionen Enable, SetCheck und GetHandled verwenden werden, sind hier alle Funktionen von TCommandEnabler beschrieben, um deutlich zu machen, wie ObjectWindows Befehle verarbeitet. Die folgenden Abschnitte beschreiben die dynamischen Vorgänge der Befehlsverarbeitung.

Behandlung von Befehlsbotschaften

Befehle sind Botschaften vom Windows Typ WM_COMMAND, mit denen Befehlsbezeichner assoziiert sind (beispielsweise CM_FILEMENU). Immer wenn der Benutzer ein Element aus einem Menü oder aus einer Mauspalette auswählt, wenn ein Dialogelement eine Benachrichtigungsbotschaft an sein übergeordnetes Fenster sendet oder wenn ein Tastenkürzel übersetzt wird, wird eine WM_COMMAND-Botschaft an ein Fenster geschickt.

Reaktion auf Befehlsbotschaften

Je nach Typ werden Befehle unterschiedlich behandelt. Für Menüeinträge und Tastenkürzelbefehle wird einer Botschaftsantworttabelle ein Befehlseintrag mit dem Makro EV_COMMAND hinzugefügt. Dieser Eintrag benötigt zwei Argumente:

- Einen Befehlsbezeichner (z.B. CM_LISTUNDO)
- Eine Elementfunktion (z.B. CMEditUndo)

Benachrichtigungen mit der Kennung untergeordneter Fenster, also Botschaften, die ein untergeordnetes Fenster an sein übergeordnetes Fenster schickt, werden mit einem der Benachrichtigungsmakros behandelt, die in der Header-Datei windoev.h definiert sind.

Benachrichtigungen mit der Kennung untergeordneter Fenster können auch vom untergeordneten Fenster selbst verarbeitet werden, wenn nämlich der Botschaftsantworttabelle des untergeordneten Fensters mit dem Makro EV_NOTIFY_AT_CHILD ein Eintrag hinzugefügt wird. Dieser Eintrag benötigt die folgenden Argumente:

- Eine Benachrichtigungsbotschaft (z.B. LBN_BDLCLK)
- Eine Elementfunktion (z.B. CmEditItem)

Befehlsverarbeitung in TWindow

TWindow ist eine der Klassen für die Befehlsverarbeitung. Es führt die Befehlsverarbeitung im Wesentlichen wie folgt durch:

Die Elementfunktion WindowProc ruft die virtuelle Elementfunktion EvCommand auf.

EvCommand überprüft, ob das Fenster die Verarbeitung des Befehls angefordert hat, und schlägt zu diesem Zweck den Befehl in der Botschafts-Antworttabelle nach.

Wenn das Fenster die Verarbeitung des Befehlsbezeichners mit dem Makro EV_COMMAND angefordert hat, wird der Befehl zugeteilt.

TWindow ist ebenfalls für die Verarbeitung von Benachrichtigungen mit der Kennung untergeordneter Fenster auf der Ebene des untergeordneten Fensters zuständig.

Befehlsverarbeitung in TFrameWindow

TFrameWindow stellt eine Sonderform der Befehlsverarbeitung zur Verfügung, indem es seine Elementfunktion EvCommand überschreibt und den Befehl die Befehlskette hinunterschickt (also die Kette der Fenster vom fokussierten Fenster zurück bis zum Rahmenfenster, dem ursprünglichen Empfänger der Befehlsbotschaft).

Wenn der Befehl von keinem der Fenster in der Befehlskette verarbeitet wird, delegiert TFrameWindow ihn an das Anwendungsobjekt. Dieser letzte Schritt wird theoretisch vom Rahmenfenster ausgeführt, tatsächlich übernimmt ihn aber die Elementfunktion DefaultProcessing von TWindow.

EvCommand aufrufen

Wenn TFrameWindow einen Befehl die Befehlskette hinunterschickt, teilt es den Befehl nicht direkt zu. Statt dessen ruft es die Elementfunktion EvCommand des Fensters auf. Dieses Vorgehen gibt den Fenstern in der Befehlskette die Möglichkeit, einen Befehl durch Überschreiben der Elementfunktion EvCommand zu verarbeiten, statt sich auf die Verarbeitung von Befehlen beschränken zu müssen, die von dem Makro EV_COMMAND angefordert wurden.

Verarbeitung von Befehlsaktivierungsbotschaften

Die meisten Anwendungen verwenden einen erheblichen Aufwand auf das Aktualisieren von Menüelementen und Mauspalettenschaltern. Dies ist erforderlich, damit die Aktivierung eines Befehls zurückgemeldet und sodann auch angezeigt wird.Um diese Vorgänge zu vereinfachen, läßt ObjectWindows die Befehlsverarbeitungsroutine, die einen Befehl verarbeiten wird, auch die Entscheidung treffen, ob der Befehl aktiviert oder deaktiviert wird.

Trotzdem durchläuft die Botschaft WM_COMMAND_ENABLE die gleiche Befehlskette wie das Ereignis WM_COMMAND. Wann genau die Botschaft WM_COMMAND_ENABLE gesendet wird, hängt vom Typ der zur Verarbeitung anstehenden Befehlsaktivierung ab.

Befehlsaktivierung für Menüeinträge

TFrameWindow führt eine Befehlsaktivierung dieses Typs durch, wenn es eine Botschaft WM_INITMENUPOPUP erhält. Es schickt diese Botschaft, bevor eine Menüliste angezeigt wird. ObjectWindows identifiziert dann die Menübefehle anhand der Befehlskennungen und fordert die Aktivierung der Befehle an.

ObjectWindows verwaltet Mauspaletten und Menüelemente dynamisch. Daher ist zu beachten, daß alle Änderungen, die an Variablen der Funktionen zur Befehlsaktivierung vorgenommen werden, dynamisch implementiert werden, nicht etwa nur bei der Aktivierung eines Fensters.

Befehlsaktivierung für Mauspalettenschalter

Dieser Typ von Befehlsaktivierung wird während der Idle-Verarbeitung durchgeführt (in der Funktion IdleAction). In dem Diagramm, das auf die Beschreibung von TWindow::DefaultProcessing folgt, ist dieser Vorgang grafisch veranschaulicht.

Erzeugen spezialisierter Befehlsaktivierer

Mit der Botschaft WM_COMMAND_ENABLE ist ein Objekt des Typs TCommandEnabler assoziiert. Diese Familie von Befehlsaktivierern umfaßt spezialisierte Befehlsaktivierer für Menüelemente und Mauspalettenschalter.

Wie aus der Klassendeklaration von TCommandEnabler zu ersehen ist, gibt es bei Verwendung des Befehlsaktivierers erheblich mehr Möglichkeiten als das bloße Aktivieren oder Deaktivieren eines Befehls. Beispielsweise läßt sich der mit einem Befehl verbundene Text ebenso ändern wie der Status des Befehls.

Verwendung des Makros EV_COMMAND_ENABLE

Sie können das Makro EV_COMMAND_ENABLE verwenden, um Botschaften des Typs WM_COMMAND_ENABLE zu verarbeiten. Genau wie auch bei dem Makro EV_COMMAND geben Sie den Bezeichner des

Befehls an, den Sie verarbeiten möchten und die Elementfunktion, die Sie zur Verarbeitung des Befehls aufrufen möchten.

Automatisches Aktivieren und Deaktivieren von Befehlen

ObjectWindows vereinfacht das Aktivieren und Deaktivieren von Befehlen, indem es automatisch alle Befehle deaktiviert, mit denen keine Behandlunsroutinen assoziiert sind. Die Elementfunktion EvCommandEnable von TWindow führt diese Operation durch, die die zweimalige Abarbeitung eines Algorithmus mit sich bringt.

Im ersten Durchgang durchläuft eine Botschaft WM_COMMAND_ENABLE die Befehlskette und so wird jedem Fenster Gelegenheit gegeben, die Befehlsaktivierung vorzunehmen.

Wenn die Anforderung zur Befehlsaktivierung von keinem der Fenster verarbeitet wird, überprüft ObjectWindows, ob eines der Fenster in der Befehlskette den Befehl über einen assoziierten Eintrag EV_COMMAND in seiner Antworttabelle verarbeiten wird. Findet sich eine Befehlsbehandlungsroutine in einer der Antworttabellen, wird der Befehl aktiviert; andernfalls wird er deaktiviert.

Aufgrund dieser eingebauten Befehlsaktivierung und -deaktivierung brauchen Sie keine ausdrückliche Befehlsaktivierung vorzunehmen. Sie sollten darauf nach Möglichkeit verzichten, sofern Sie nicht den Befehlstext oder den Befehlsstatus ändern möchten oder einen Befehl bedingt aktivieren oder deaktivieren möchten.

Wenn Sie Befehle indirekt durch Überschreiben der Elementfunktion EvCommand verarbeiten, statt über das Makro EV_COMMAND einen Antworttabelleneintrag hinzuzufügen, dann erhält ObjectWindows von der Befehlsverarbeitung keine Kenntnis. Die mögliche Folge ist eine automatische Deaktivierung des Befehls. Sollte dieser Fall eintreten, ist die geeignete Maßnahme das Überschreiben der Elementfunktion EvCommandEnable und das ausdrückliche Aktivieren des Befehls.

Öffentliche Konstruktoren

Konstruktor

```
TCommandEnabler(uint id, HWND hWndReceiver = 0);
```

Konstruiert das Objekt TCommandEnable mit der angegebenen Befehlskennung. Setzt den Empfänger der Botschaft (hWndReceiver) auf 0.

Typdefinitionen

Check State

```
enum CheckState{Unchecked, Checked, Indeterminate};
```

Zählt die Werte für den Markierungszustand des Befehlssenders auf. Dieser Zustand bezieht sich auf Schalter, wie sie in Mauspaletten oder als Gadgets in Steuerbalken vorkommen.

Öffentliche Datenelemente

Id

```
const uint Id;
```

Befehlskennung für den aktivierten Befehl.

Öffentliche Elementfunktionen

CommandEnabler

Enable

```
virtual void Enable(bool enable = true);
```

Aktiviert oder deaktiviert den Befehlssender. Wenn Enable aufgerufen wird, setzt es das Flag Handled.

GetHandled

```
bool GetHandled();
```

Gibt Handled zurück, einen Flag-Wert, der anzeigt, ob der Befehlsaktivierer die Verarbeitung übernommen hat. In diesem Fall ist Handled true.

IsReceiver

```
bool IsReceiver(HWND hReceiver);
```

Gibt true zurück, wenn receiver der ursprünglich im Konstruktor festgelegte Empfänger der Botschaft ist.

SetCheck

```
virtual void SetCheck(int check) = 0;
```

Ändert den Markierungszustand des Befehlssenders in markiert, unmarkiert oder unbestimmt. Dieser Zustand bezieht sich auf Schalter, wie sie in Mauspaletten oder als Gadgets in Steuerbalken vorkommen.

SetText

```
virtual void SetText(const char far* text) = 0;
```

Ändert den mit einem Befehlssender verbundenen Text. Das kann, beispielsweise, Text sein, der mit einem Menüelement verbunden ist, oder Text auf einem Schalter.

Geschützte Datenelemente

Handled

```
bool Handled;
```

Ist true, wenn der Befehlsaktivierer verarbeitet worden ist.

HWndReceiver

```
const HWND HWndReceiver;
```

Der Botschaftsempfänger (das Fenster), das den Befehl erhält.

Klasse TCommonDialog — commdial.h

Als eine von TDialog abgeleitete Klasse stellt TCommonDialog die abstrakte Basisklasse für TCommonDialog-Objekte dar. Sie bietet die grundlegende Funktionalität für das Erzeugen von Dialogfenstern, wobei auf

die DLL für allgemeine Dialogfenster zurückgegriffen wird. Zu den Klassen von ObjectWindows, die diese Funktionalität allgemeiner Dialogfenster erben, gehören

TChooseColorDialog - ein modales Dialogfenster, das den Benutzer die Farben für eine Anwendung auswählen läßt.

TChooseFontDialog - ein modales Dialogfenster, das den Benutzer die Schriftart für eine Anwendung auswählen läßt.

TReplaceDialog - ein nicht-modales Dialogfenster, das den Benutzer eine Textauswahl, die ersetzt werden soll, festlegen läßt.

TFindDialog - ein nicht-modales Dialogfenster, das den Benutzer eine Textauswahl, nach der gesucht werden soll, festlegen läßt.

TFileOpenDialog - ein modales Dialogfenster, in dem der Benutzer eine zu öffnende Datei angeben kann.

TFileSaveDialog - ein modales Dialogfenster, in dem der Benutzer eine zu speichernde Datei angeben kann.

TPrintDialog - ein modales Dialogfenster, in dem der Benutzer Druckeroptionen festlegen kann.

Jede allgemeine Dialogklasse verwendet eine eingebettete Klasse, TData, die die Attribute und die vom Benutzer festgelegten Daten speichert. Beispielsweise speichert die Klasse TChooseColorDialog::TData die Farbattribute, die der Benutzer als Antwort auf eine Abfrage in einem allgemeinen Dialogfenster auswählt. Tatsächlich wird beim Konstruieren eines allgemeinen Dialogfensters zuerst ein Objekt TData konstruiert und seinen Datenelementen werden Werte zugewiesen, und erst danach kann ein allgemeines Dialogfensterobjekt konstruiert werden. Schließlich wird entweder ein modales oder ein nicht-modales Dialogfenster erzeugt und ausgeführt.

Öffentliche Konstruktoren

Konstruktor

```
TCommonDialog(TWindow* parent, const char far* title = 0,
              TModule* module = 0);
```

Ruft einen TWindow-Konstruktor auf, übergibt das übergeordnete Fenster parent, und konstruiert dann ein allgemeines Dialogfenster, das Sie nach Ihren Anforderungen verändern können. Sie können die Modulkennung (module) und den Fenstertitel (title) festlegen. Die voreingestellten Werte sind 0.

Öffentliche Elementfunktionen

DoCreate

```
HWND DoCreate();
```

Wird DoCreate von Create aufgerufen, erzeugt sie ein nicht-modales Dialogfenster. Sie gibt bei Mißerfolg 0 zurück.

Siehe auch: TDialog::Create

DoExecute

```
int DoExecute();
```

Wird DoExecute von Execute aufgerufen, erzeugt sie ein modales Dialogfenster. Sie gibt IDCANCEL zurück, wenn der Vorgang abgebrochen wurde oder nicht erfolgreich war.

Siehe auch: TDialog::Execute

Geschützte Datenelemente

CDTitle

```
const char far* CDTitle;
```

CDTitle speichert den optionalen Titel, der im allgemeinen Dialogfenster angezeigt wird.

Siehe auch: TDialog::SetCaption

Geschützte Elementfunktionen

CmHelp

```
void CmHelp();
```

Standardbehandlungsroutine für den Aktionsschalter pshHelp (den Schalter »Hilfe« im Dialogfenster) .

CmOkCancel

```
void CmOkCancel();
```

Antwortet auf ein Klicken auf den »OK«- oder »Abbruch«-Schalter des Dialogfensters, indem DefaultProcessing aufgerufen wird, damit die DLL für allgemeine Dialogfenster den Befehl verarbeiten kann.

Siehe auch: TDialog::Cancel, TDialog::CmOk

EvClose

```
void EvClose();
```

Antwortet auf die Botschaft WM_CLOSE, indem DefaultProcessing aufgerufen wird, damit die DLL für allgemeine Dialogfenster den Befehl verarbeiten kann.

Siehe auch: TDialog::EvClose

SetupWindow

```
void SetupWindow();
```

Weist den Titel des Dialogfensters CDTitle zu, wenn CDTitle Nicht-Null ist.

Siehe auch: TDialog::SetupWindow

Antworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_COMMAND(IDCANCEL, CmOkCancel)	CmOkCancel
EV_COMMAND(IDOK, CmOkCancel)	CmOkCancel
EV_WM_CLOSE	EvClose
EV_WM_CTLCOLOR	EvCtlColor

Typdefinition TCondFunc — window.h

```
typedef bool (*TCondFunc) (TWindow *win, void *param);
```

Definiert einen Elementfunktionstyp, der von der Funktion FirstThat von TWindow benutzt wird .

Siehe auch: TWindow::FirstThat

Typdefinition TCondMemFunc — window.h

```
typedef bool (TWindow::TCondMemFunc) (*win, void *param);
```

Definiert einen Elementfunktionstyp, der von der Funktion FirstThat von TWindow benutzt wird.

Siehe auch: TWindow::FirstThat

Klasse TControlBar — controlb.h

Als eine von TGadgetWindow abgeleitete Klasse implementiert TControlBar eine Dialogelementleiste, die mnemonischen Zugriff auf ihre Schalter-Gadgets bietet. Um eine Dialogelementleiste in einem Rahmenfenster zu konstruieren, zu erzeugen und einzufügen, können Sie zuerst die folgende Antworttabelle definieren:

```
DEFINE_RESPONSE_TABLE1(TMDIFileApp, TApplication)
  EV_COMMAND(CM_FILENEW, CmFileNew),
  EV_COMMAND(CM_FILEOPEN, CmFileOpen),
  EV_COMMAND(CM_SAVESTATE, CmSaveState),
  EV_COMMAND(CM_RESTORESTATE, CmRestoreState),
END_RESPONSE_TABLE;
```

Fügen Sie als nächstes Anweisungen hinzu, die ein Hauptfenster konstruieren, und laden Sie dessen Menü, Tastenkürzeltabelle und Symbol. Fügen Sie dann die folgenden Anweisungen ein, um eine Dialogelementleiste zu konstruieren, zu erzeugen und sie in das Rahmenfenster einzufügen:

```
TControlBar* cb = new TControlBar(frame);
cb->Insert(*new TButtonGadget(CM_FILENEW, CM_FILENEW));
cb->Insert(*new TButtonGadget(CM_FILEOPEN, CM_FILEOPEN));
cb->Insert(*new TButtonGadget(CM_FILESAVE, CM_FILESAVE));
cb->Insert(*new TSeparatorGadget(6));
cb->Insert(*new TButtonGadget(CM_EDITCUT, CM_EDITCUT));
cb->Insert(*new TButtonGadget(CM_EDITCOPY, CM_EDITCOPY));
cb->Insert(*new TButtonGadget(CM_EDITPASTE, CM_EDITPASTE));
cb->Insert(*new TSeparatorGadget(6));
cb->Insert(*new TButtonGadget(CM_EDITUNDO, CM_EDITUNDO));
frame->Insert(*cb, TDecoratedFrame::Top);
```

Das ObjectWindows-Beispielprogramm MDIFILE.CPP auf Ihrer mitgelieferten Diskette zeigt eine Dialogelementleiste auf dem Bildschirm an.

Öffentliche Konstruktoren

Konstruktor

```
TControlBar(TWindow* parent = 0, TTileDirection direction = Horizontal,
            TFont* font = new TGadgetWindowFont, TModule* module = 0);
```

Konstruiert ein TControlBar-Oberflächenobjekt mit der angegebenen Richtung (horizontal oder vertikal) und der Fensterschrift.

Öffentliche Elementfunktionen

PreProcessMsg

```
bool PreProcessMsg(MSG& msg);
```

Führt die Vorverarbeitung von Windows-Botschaften durch. Da PreProcessMsg keine Tastenkürzel für TControlBar übersetzt, gibt die Funktion false zurück.

Geschützte Elementfunktionen

PositionGadget

```
void PositionGadget(TGadget* previous, TGadget* next, TPoint& p);
```

Erhält die Rahmeneigenschaft, bestimmt die Richtung des Gadget und positioniert das Schalter-Gadget entweder auf einem horizontalen oder einem vertikalen Rahmen, wenn irgendeine Überlappung erforderlich ist.

Klasse TControlGadget controlg.h

TControlGadget dient als ein Ersatz für TControl, so daß Sie TControl-Objekte wie Editierfelder, Schalter, Gleitregler, Meßanzeigen oder VBX-Dialogelemente in ein Gadget-Fenster plazieren können. Falls notwendig, setzt TControlGadget ein übergeordnetes Fenster und erzeugt das Steuer-Gadget. Siehe TGadget für weitere Informationen zu Gadgetobjekten.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TControlGadget(TWindow& control, TBorderStyle = None);
```

Erzeugt ein TControlGadget-Objekt, das mit dem angegebenen TControl-Fenster assoziiert ist.

Destruktor

```
~TControlGadget();
```

Löscht ein TControlGadget-Objekt und entfernt es aus dem assoziierten Fenster.

Geschützte Datenelemente

Control

```
TWindow* Control;
```

Zeigt auf das Dialogelementfenster, das durch TControlGadget verwaltet wird.

Geschützte Elementfunktionen

GetDesiredSize

```
void GetDesiredSize(TSize& size);
```

Ruft TGadget::GetDesiredSize auf und übergibt die Größe size dem Steuer-Gadget. Verwenden Sie GetDesiredSize, um die Größe des Steuer-Gadgets zu ermitteln. Seine Größe muß sowohl Rahmen und Ränder als auch die größte vorkommende Breite und die größte vorkommende Höhe aufnehmen können.

Siehe auch: TGadget::GetDesiredSize

GetInnerRect

```
void GetInnerRect(TRect&);
```

Berechnet den Rechteckbereich des Steuer-Gadget ohne Rahmen und Ränder.

Inserted

```
void Inserted();
```

Wird aufgerufen, wenn das Steuer-Gadget in das übergeordnete Fenster eingefügt wird. Zeigt das Fenster in seiner aktuellen Größe und Position an.

Invalidate

```
void Invalidate(bool erase = true);
```

Wird benutzt, um den aktiven Teil (normalerweise ohne Rahmen) des Steuer-Gadget ungültig zu machen. Invalidate ruft InvalidateRect auf und übergibt die Breite und Höhe des freizugebenden Bereiches.

InvalidateRect

```
void InvalidateRect(const TRect&, bool erase = true);
```

Macht das Rechteck des Steuer-Gadget im übergeordneten Fenster ungültig.

Removed

```
void Removed();
```

Wird aufgerufen, wenn das Steuer-Gadget vom übergeordneten Fenster entfernt wird.

SetBounds

```
void SetBounds(TRect& rect);
```

Ruft TGadget::SetBounds auf und die Dimensionen des Rechtecks des Steuer-Gadget. SetBounds informiert das Steuer-Gadget über eine Änderung innerhalb ihres begrenzenden Rechteckes.

Siehe auch: TGadget::SetBounds

Update

```
void Update();
```

Aktualisiert den Client-Bereich des angegebenen Fensters, indem sofort eine WM_PAINT-Botschaft gesendet wird.

Antworttabelleneinträge

Die Klasse TControlGadget hat keine Einträge in der Antworttabelle.

Klasse TControl — control.h

Die Klasse TControl vereint ihre abgeleiteten Klassen, wie zum Beispiel TScrollBar, TControlGadget und TButton. Dialogelementobjekte abgeleiteter Klassen werden zur Repräsentation von Dialogelementen als Oberflächenelemente in Windows benutzt. Zur Erzeugung eines Dialogelementes in einem übergeordneten TWindow oder einem abgeleiteten Fenster muß ein Dialogelementobjekt verwendet werden. Ein Dialogelementobjekt kann zur Erleichterung der Kommunikation zwischen Ihrer Anwendung und den Dialogelementen eines TDialog dienen. TControl ist eine streamfähige Klasse.

Öffentliche Konstruktoren

Konstruktor

```
TControl(TWindow *parent, int Id, const char far *title, int x, int y,
        int w, int h, TModule* module = 0);
```

Ruft den Konstruktor von TWindow auf und übergibt ihm parent (übergeordnetes Fenster), title (Titeltext) und module. Setzt Attr mit Hilfe der mitgelieferten Parameter Bibliotheks-ID (Id), Position (x, y) relativ zum Ursprung des Client-Bereichs des übergeordneten Fensters, der Breite (w) und der Höhe (h). Sie setzt ferner Attr.Style auf WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP.

Siehe auch: TWindow::TWindow

Konstruktor

```
TControl(TWindow *parent, int resourceId, TModule* module = 0);
```

Konstruiert ein Objekt, das mit einem Oberflächendialogelement von TDialog assoziiert werden soll. Ruft den Konstruktor von TWindow auf und aktiviert den Datentransfermechanismus über den Aufruf von EnableTransfer.

Der Parameter ResId muß mit einer Oberflächen-Ressource für ein Dialog-element übereinstimmen, die Sie definieren.

Siehe auch: TWindow::TWindow, TWindow::EnableTransfer

Öffentliche Elementfunktionen

CompareItem

```
virtual int CompareItem (COMPAREITEMSTRUCT far &);
```

Wenn CompareItem in Kombinationsfenstern verwendet wird, die durch ein übergeordnetes Fenster gezeichnet werden, vergleicht die Funktion zwei Elemente. Die abgeleitete Klasse liefert die Vergleichslogik.

DeleteItem

```
virtual void DeleteItem (DELETEITEMSTRUCT far &);
```

DeleteItem wird in Kombinationsfenstern benutzt, die durch ein übergeordnetes Fenster gezeichnet werden. In solchen Fällen liefert die abgeleitete Klasse die Logik zum löschen.

DrawItem

```
virtual void DrawItem(DRAWITEMSTRUCT far &drawInfo);
```

DrawItem antwortet auf eine Botschaft, die durch TWindow an ein zeichenbares Dialogelement weitergereicht wurde, wenn das Dialogelement gezeichnet werden muß. TControl::DrawItem ruft ODADrawEntire auf, wenn das Dialogelement vollständig gezeichnet werden muß, sie ruft ODASelect auf, wenn sich der Auswahlzustand des Dialogelements geändert hat, oder sie ruft ODAFocus auf, wenn der Fokus vom oder zum Dialogelement umgeschaltet wurde.

Siehe auch: TControl::ODADrawEntire, TControl::ODASelect, TControl::ODAFocus, TWindow::EvDrawItem

MeasureItem

```
virtual void MeasureItem (MEASUREITEMSTRUCT far &);
```

Diese Funktion wird von Dialogelementen verwendet, die durch ein übergeordnetes Fenster gezeichnet werden, um Windows über die Dimensionen des angegebenen Elements zu informieren. Bei Listenfeldern und Dialogelementfeldern bezieht sich diese Funktion auf spezifische Elemente. Bei anderen Dialogelementen, die durch ein übergeordnetes Fenster gezeichnet werden, wird die Funktion benutzt, um Windows über die absolute Größe des Dialogelements zu informieren.

Geschützte Elementfunktionen

EvPaint

```
void EvPaint();
```

Wenn das Dialogelement eine vordefinierte Windows-Klasse besitzt, ruft EvPaint die Funktion TWindow::DefaultProcesing für das von Windows bereitgestellte Zeichnen auf. Anderenfalls ruft sie TWindow::EvPaint auf.

Siehe auch: TWindow::DefaultProcesing, TWindow::EvPaint

ODADrawEntire

```
virtual void ODADrawEntire(DRAWITEMSTRUCT far &drawInfo);
```

Antwortet auf eine Benachrichtigungsbotschaft, die an ein zeichenbares Dialogelement gesendet wird, wenn eine Zeichnung des Dialogelementes erforderlich ist. ODADrawEntire kann durch ein zeichenbares Dialogelement redefiniert werden, um die Art seiner Zeichnung festzulegen.

Siehe auch: TWindow::EvDrawItem , TControl::DrawItem

ODAFocus

```
virtual void ODAFocus(DRAWITEMSTRUCT far &drawInfo);
```

Antwortet auf eine Benachrichtigung, die an ein zeichenbares Dialogelement gesendet wird, wenn der Fokus vom oder zum Dialogelement umgeschaltet wurde. ODAFocus kann durch ein zeichenbares Dialogelement redefiniert werden, um die Art festzulegen, wie es im Falle des Fokusverlustes oder -gewinnes gezeichnet werden soll.

Siehe auch: TWindow::EvDrawItem , TControl::DrawItem

ODASelect

```
virtual void ODASelect(DRAWITEMSTRUCT far &drawInfo);
```

Antwortet auf eine Benachrichtigung, die an ein zeichenbares Dialogelement gesendet wird, wenn sich der Auswahlzustand des Dialogelements geändert hat. Standardmäßig ruft ODASelect Parent->DrawItem auf. ODASelect kann durch ein zeichenbares Dialogelement redefiniert werden, um die Art festzulegen, wie es gezeichnet werden soll, wenn sich der Auswahlzustand geändert hat.

Siehe auch: TWindow::EvDrawItem, DrawItem

Antworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_WM_PAINT	EvPaint

Klasse TCreatedDC dc.h

Die abstrakte Klasse TDC mit dem Namen TCreatedDC dient als Basis für DCs, die erzeugt und gelöscht werden. TCreatedDC übernimmt einen großen Teil der Arbeit für das Erzeugen und Löschen von HDCs. Sie verwendet dabei die Funktionen CreateDC und ::DeleteDC. Siehe TDC für weitere Informationen zu DC-Objekten.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TCreatedDC(const char far* driver, const char far* device,
           const char far* output, const DEVMODE far* initData=0);
```

Erzeugt ein DC-Objekt für das Gerät, welches angegeben wird durch driver (den Treibernamen), device (den Gerätenamen) und output (den Namen der DOS-Datei oder des Gerätes [des Ports] für das physikalische Ausgabemedium). Das optionale Argument initData stellt eine DEVMODE-Struktur bereit, die gerätespezifische Initialisierungsdaten für diesen DC enthält. initData muß 0 (der Vorgabewert) sein, wenn das Gerät irgendwelche Standardinitialisierungen verwenden soll, die durch den Benutzer über die Systemsteuerung angegeben wurden.

Siehe auch: ::CreateDC, ::ExtDeviceMode

Konstruktor

```
TCreatedDC(HDC handle, TAutoDelete autoDelete);
```

Erzeugt ein DC-Objekt mit Hilfe eines existierenden DC.

Siehe auch: enum TDC::TAutoDelete, TDC::ShouldDelete

Destruktor

```
~TCreatedDC();
```

Ruft RestoreObjects auf und löscht alle OrgXXX-Datenelemente, die nicht gleich Null sind. Wenn ShouldDelete true ist, löscht der Destruktor diesen DC über ::DeleteDC.

Siehe auch: TDC::ShouldDelete, ::DeleteDC, TDC::RestoreObjects

Geschützte Elementfunktionen

TCreatedDC

```
TCreatedDC();
```

Erzeugt einen Gerätekontext für das gegebene Gerät. DC-Objekte können konstruiert werden, indem ein existierendes HDC geliehen wird oder indem Geräte- und Treiberinformationen bereitgestellt werden.

Siehe auch: ::CreateDC

Klasse TCursor gdiobjec.h

TCursor ist von TGdiBase abgeleitet und repräsentiert die GDI-Mauszeiger-Objektklasse. Konstruktoren von TCursor können Mauszeiger von einer Ressource oder von einer expliziten Information aus erzeugen. Da Mauszeiger keine wirklichen GDI-Objekte darstellen, überlädt der Destruktor von TCursor den Basisdestruktor ~TGdiBase().

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TCursor(HCURSOR handle, TAutoDelete autoDelete = NoAutoDelete);
```

Erzeugt ein TCursor-Objekt und setzt das Datenelement Handle auf das gegebene geborgte handle. Das Datenelement ShouldDelete hat standardmäßig den Wert false. Dadurch wird sichergestellt, daß das geborgte Handle nicht gelöscht wird, wenn das C++ Objekt entfernt wird.

Konstruktor

```
TCursor(HINSTANCE instance, const TCursor& cursor);
```

Erzeugt eine Kopie des gegebenen cursor-Objektes. Die 32-Bit-Version (zum Compilieren einer Win32-Anwendung) verwendet CopyIcon() und führt eine Typumwandlung durch, um auf HICON zuzugreifen.

Konstruktor

```
TCursor(HINSTANCE instance, TResID resID);
```

Konstruiert ein cursor-Objekt aus der gegebenen Ressourcen-ID.

Konstruktor

```
TCursor(HINSTANCE instance, const TPoint& hotSpot,
        const TSize& size, void far* andBits, void far* xorBits);
```

Konstruiert ein TCursor-Objekt mit der angegeben Größe an dem angegebenen Punkt.

Konstruktor

```
TCursor(const void* resBits, uint32 resSize);
```

Nur 32-Bit. Konstruiert ein TCursor-Objekt aus der angegebenen Ressource.

Konstruktor

```
TCursor(const ICONINFO* iconInfo);
```

Erzeugt ein TCursor-Objekt aus der angegebenen Strukturinformation ICONINFO.

Siehe auch: ::CreateIconIndirect

Destruktor

```
~TCursor();
```

Entfernt ein TCursor-Objekt.

Siehe auch: ~TGdiObject, TGdiObject::Handle, TGdiObject::ShouldDelete, TPoint, TSize, Struktur ICONINFO

Öffentliche Elementfunktionen

GetIconInfo

```
bool GetIconInfo(ICONINFO* iconInfo) const;
```

Fragt Informationen über dieses Symbol ab und kopiert sie in die angegebene Struktur ICONINFO. Gibt true zurück, wenn der Aufruf erfolgreich ist; andernfalls false.

Siehe auch: Struktur ICONINFO

Operatoren

Operator HCURSOR()

```
operator HCURSOR() const;
```

Ein Inline-Typumwandlungsoperator. Konvertiert das Handle dieses Mauszeigers in den Typ HCURSOR (den Windows-Datentyp, der das Handle auf eine Mauszeiger-Ressource repräsentiert).

Operator ==

```
bool operator ==(const TCursor& other) const;
```

Gibt true zurück, wenn dieser Cursor other gleicht; gibt andernfalls false zurück.

Klasse TDC — dc.h

TDC ist die Stammklasse für GDI-DC-Rahmenobjekte . Jedes *TDC*-Objekt besitzt ein *Handle* als geschütztes Datenelement vom Typ HDC (Handle auf DC). Win-API-Funktionen, die ein HDC als Argument erwarten, können deshalb über eine korrespondierende *TDC*-Elementfunktion ohne dieses explizite Handle-Argument aufgerufen werden.

DC-Objekte können mit Konstruktoren von *TDC* direkt oder über Konstruktoren von spezialisierten Subklassen (wie zum Beispiel *TWindowDC*, *TMemoryDC*, *TMetaFileDC*, *TDibDC* und *TPrintDC*) erzeugt werden, um spezifisches Verhalten zu erhalten. DC-Objekte können mit einem bereits existierenden oder entliehenen HDC-Handle oder ausgehend von Geräte-

und Treiberinformationen wie bei *::CreateDC()* erzeugt werden. Die Klasse *TCreateDC* übernimmt einen großen Teil der Arbeit von *TDC* für das Erzeugen und Löschen.

TDC besitzt vier Handles als geschützte Datenelemente: *OrgBrush*, *OrgPen*, *OrgFont* und *OrgPalette*. Diese Handles verwalten die Standard-GDI-Objekte, die innerhalb jedem DC ausgewählt sind. Sobald neue GDI-Objekte mit *SelectObject()* oder *SelectPalette()* ausgewählt sind, speichern diese Datenelemente die vorherigen Objekte. Die letzteren können einzeln über *RestoreBrush*, *RestorePen()* usw. oder insgesamt mit *RestoreObjects*() wieder eingesetzt werden. Wenn ein *TDC*-Objekt entfernt wird (über *~TDC::TDC*), werden alle ursprünglich ausgewählten Objekte wieder eingesetzt. Das Datenelement *TDC::ShouldDelete* kontrolliert das Löschen des zugrundeliegenden TDC-Objekts.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TDC(HDC handle);
```

Erzeugt ein DC-Objekt, indem es das Handle eines existierenden DC »borgt«. Das Datenelement *Handle* wird auf das gegebene Argument *handle* gesetzt.

Destruktor

```
virtual ~TDC();
```

Ruft *RestoreObjects* auf.

Siehe auch: *TCreateDC, TDC::ShouldDelete, TDC::RestoreObjects*

Öffentliche Elementfunktionen

AngleArc

```
bool AngleArc(int x, int y, uint32 radius, float startAngle,
              float sweepAngle);

bool AngleArc(const TPoint& center, uint32 radius, float
              startAngle, float sweepAngle);
```

Nur 32-Bit. Zeichnet ein Liniensegment und einen Bogen auf diesem DC mit Hilfe des aktuell ausgewählten Stiftobjektes. Die Linie wird von der aktuellen Position bis zum Beginn des Bogens gezogen. Der Bogen ist derjenige Teil des Kreises (mit dem Mittelpunkt bei den logischen Koordinaten (*x*, *y*) und dem positivem Radius *radius*), der bei *startAngle* startet und bei
(*startAngle* + *sweepAngle*) endet. Beide Winkel sind von der x-Achse aus in Grad gegen den Uhrzeigersinn (die Standardrichtung für Bögen) gemessen. Der Bogen könnte elliptisch erscheinen. Dies hängt vom aktuellen Transformations- und Abbildungsmodus ab. *AngleArc* gibt TRUE zurück, wenn die Figur erfolgreich gezeichnet wurde. Andernfalls gibt sie

FALSE zurück. Bei Erfolg hat sich die aktuelle Position an den Endpunkt des Bogens bewegt.

Siehe auch: *TDC::Arc, Klasse TPoint*

Arc

```
bool Arc(int x1, int y1, int x2, int y2, int x3, int y3,
                int x4, int y4);

bool Arc(const TRect& r, const TPoint& start, const TPoint& end);
```

Zeichnet einen elliptischen Bogen auf diesen DC mit Hilfe des aktuell ausgewählten Stiftobjektes. Der Mittelpunkt des Bogens ist der Mittelpunkt des umgebenden Rechtecks, das entweder durch (*x1, y1*)/(*x2, y2*) oder durch das Rechteck *r* angegeben ist. Die Start- und Endpunkte des Bogens sind entweder durch (*x3, y3*)/(*x4, y4*) oder durch die Punkte *start* und *end* angegeben. Alle Punkte werden in logischen Koordinaten angegeben. *Arc* gibt TRUE zurück, wenn der Bogen erfolgreich gezeichnet wurde. Andernfalls gibt sie FALSE zurück. Die aktuelle Position wird bei diesem Aufruf weder benutzt noch verändert. Die Zeichenrichtung wird durch *::SetArcDirection* gesetzt; die Standardrichtung ist gegen den Uhrzeigersinn.

Siehe auch: *TPoint, TRect, TDC::AngleArc*

BeginPath

```
bool BeginPath();
```

Nur 32-Bit. Öffnet eine neue Pfadsammlung für diesen DC und verwirft alle vorherigen Pfade von diesem DC. Wenn eine Pfadsammlung einmal geöffnet ist, kann eine Anwendung mit dem Aufruf von Zeichenfunktionen auf diesem DC beginnen, um die Punkte zu definieren, die innerhalb dieses Pfades liegen. Die Zeichenfunktionen, die Punkte in einem Pfad definieren sind die folgenden *TDC*-Elemente: *AngleArc, Arc, Chord, CloseFigure,*
Ellipse, ExtTextOut, LineTo, MoveToEx, Pie, PolyBezier, PolyBezierTo, PolyDraw, Polygon, Polyline, PolylineTo, PolyPolygon, PolyPolyline, Rectangle, RoundRect und *TextOut*.

Eine Pfadsammlung kann über den Aufruf von *TDC::EndPath* geschlossen werden. *BeginPath* gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::FillPath, TDC::EndPath, TDC::PathToRegion, TDC::StrokePath, TDC::StrokeandFillPath, TDC::WidenPath*

BitBlt

```
bool BitBlt(int dstX, int dstY, int w, int h, const TDC& srcDC,
                int srcX, int srcY, uint32 rop=SRCCOPY);

bool BitBlt(const TRect& dst, const TDC& srcDC, const TPoint& src,
                uint32 rop=SRCCOPY);
```

Führt einen Bitblock-Transfer von *srcDc* (dem gegebenen Quell-DC) zu diesem DC (dem Ziel-DC) durch. Farb-Bits werden von einem Quellrecht-

eck zu einem Zielrechteck kopiert. Die Position des Quellrechtecks wird entweder durch dessen linke obere Ecke in logischen Koordinaten (*srcX*, *srcY*) oder durch das *TPoint*-Objekt *src* angegeben. Das Zielrechteck kann entweder durch dessen linke obere Ecke in logischen Koordinaten (*dstX*, *dstY*), Breite *w* und Höhe *h* oder durch das *TRect*-Objekt *dst* angegeben werden. Das Zielrechteck hat dieselbe Breite und Höhe wie das Quellrechteck. Das Argument *rop* gibt die Rasteroperation an, die für die Kombination der Farbdaten für jedes Pixel benutzt wird. Siehe *TDC::MaskBlt* für eine detaillierte Liste von *rop*-Codes.

Wenn eine erweiterte Metadatei aufgezeichnet wird, tritt ein Fehler auf, sobald der Quell-DC einen erweiterten Metadatei-DC identifiziert.

Siehe auch: Klasse *TPoint*, Klasse *TRect*

Chord

```
bool Chord(int x1, int y1, int x2, int y2, int x3, int y3, int x4,
                int y4);
bool Chord(const TRect& R, const TPoint& Start, const TPoint& End);
```

Zeichnet und füllt eine Bogenfläche (einen Bereich, der durch den Schnitt zwischen einer Ellipse und einem Liniensegment begrenzt wird) auf diesem DC. Dabei werden die aktuellen Stift- und Pinselobjekte verwendet. Die Ellipse wird über ein begrenzendes Rechteck angegeben, das entweder über (*x1*, *y1*)/(*x2*, *y2*) oder über das Rechteck *R* definiert ist. Die Start- und Endpunkte der Bogenfläche sind entweder über (*x3*, *y3*)/(*x4*, *y4*) oder über die Punkte *Start* und *End* angegeben. *Chord* gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück. Die aktuelle Position wird bei diesem Aufruf weder benutzt noch verändert.

Siehe auch: *TDC::Arc*, Klasse *TPoint*, Klasse *TRect*

CloseFigure

```
bool CloseFigure();
```

Nur 32-Bit. Schließt eine offene Figur in der offenen Pfadsammlung dieses DC, indem eine Linie von der aktuellen Position bis zum ersten Punkt der Figur gezogen wird (normalerweise der Punkt, der durch den letzten Aufruf von *TDC::MoveTo* angegeben wird) und indem die Linien mit Hilfe des für den DC aktuellen Verknüpfungsstils verbunden werden. Wenn Sie eine Figur mit *TDC::LineTo* und nicht mit *CloseFigure* schließen, werden Endkappen (an Stelle von Verknüpfungen) für das Erzeugen der Ecken benutzt. Der Aufruf schlägt fehl, wenn es keine offene Pfadsammlung auf diesem DC gibt. Jede Linie oder Kurve, die dem Pfad nach einem Aufruf von *CloseFigure* zugefügt wurde, startet eine neue Figur. Eine Figur in einem Pfad bleibt offen, bis sie explizit mit *CloseFigure* geschlossen wird, auch wenn ihre aktuelle Position und ihr Startpunkt zusammenfallen.

CloseFigure gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::BeginPath, TDC::EndPath*

DPtoLP

```
bool DPtoLP(TPoint* points, int count = 1) const;
```

Konvertiert jeden der *count* Punkte im *points*-Array von Gerätepunkten in logische Punkte. Die Konvertierung hängt vom aktuellen Abbildungsmodus dieses DC und von den Einstellungen seines Fenster- und Viewport-Ursprungs sowie deren Ausdehnungen ab. *DPtoLP* gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::LPtoDP*, Klasse *TPoint*

DrawFocusRect

```
bool DrawFocusRect(int x1, int x2, int y1, int y2);
bool DrawFocusRect(const TRect& rect);
```

Zeichnet das gegebene Rechteck auf diesen DC in dem gleichen Stil, der für das Anzeigen des Fokus verwendet wird. Ein zweiter Aufruf der Funktion mit demselbem Argument *rect* wird das Rechteck von der Anzeige entfernen. Ein mit *DrawFocusRect* gezeichnetes Rechteck kann nicht geblättert werden. *DrawFocusRect* gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: Klasse *TRect*

DrawIcon

```
bool DrawIcon(int x, int y, const TIcon& icon);
bool DrawIcon(const TPoint& point, const TIcon& icon);
```

Zeichnet das gegebene *icon* auf diesen DC. Die obere linke Ecke des gezeichneten Symbols kann durch x- und y-Koordinaten oder durch das Argument *point* angegeben werden. *DrawIcon* gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: Klasse *TIcon*

DrawText

```
virtual bool DrawText(const char far* string, int count,
                      const TRect& r, uint16 format = 0);
```

Formatiert und zeichnet in dem gegebenen Rechteck *r* bis zu *count* Zeichen des nullterminierten Strings *string*. Dabei wird die aktuelle Schrift für den DC verwendet. Wenn *count* -1 ist, wird der gesamte String geschrieben. Das Rechteck muß in logischen Einheiten angegeben werden. Die Formatierung wird mit dem Argument *format* kontrolliert, das verschiedene Kombinationen der folgenden Werte annehmen kann:

Wert	Bedeutung
DT_BOTTOM	Gibt nach unten ausgerichteten Text an. Dieser Wert muß mit DT_SINGLELINE kombiniert (bitweise durch »Oder« verbunden) werden.
DT_CALCRECT	Bestimmt die Breite und Höhe des Rechtecks. Wenn mehrere Textzeilen existieren, verwendet *DrawText* die Breite von *r* (dem Rechteckargument) und erweitert die Rechteckbasis, damit die letzte Textzeile umgeben wird. Wenn nur eine Textzeile existiert, verwendet *DrawText* einen modifizierten Wert für die rechte Seite von *r*, so daß es das letzte Zeichen in der Zeile umgibt. In beiden Fällen gibt *DrawText* die Höhe des formatierten Textes zurück. Die Funktion zeichnet den Text jedoch nicht.
DT_CENTER	Zentriert Text horizontal.
DT_EXPANDTABS	Expandiert Tabulatorzeichen. Die Standardzahl von Zeichen für einen Tabulator ist acht.
DT_EXTERNALLEADING	Bindet den externen Durchschuß in die Zeilenhöhe der Schrift mit ein. Normalerweise wird der externe Durchschuß nicht in die Höhe einer Textzeile mit eingebunden.
DT_LEFT	Paßt den Text linksbündig an.
DT_NOCLIP	Zeichnet ohne Clipping. *DrawText* ist etwas schneller, wenn DT_NOCLIP nicht benutzt wird.
DT_NOPREFIX	Schaltet die Verarbeitung von Präfixzeichen ab. Normalerweise interpretiert *DrawText* das Präfixzeichen **&** als eine Anweisung, das folgende Zeichen zu unterstreichen, und das Präfixzeichen **&&** als eine Anweisung, ein einzelnes **&** zu drucken. Mit der Angabe von DT_NOPREFIX wird diese Verarbeitung abgeschaltet.
DT_RIGHT	Paßt den Text rechtsbündig an.
DT_SINGLELINE	Gibt eine Einzelzeile an. Zeilenrückläufe und Zeilenvorschübe unterbrechen die Zeilen nicht.
DT_TABSTOP	Setzt Tabs. Bits 15-8 (das höherwertige Byte des niederwertigen Wortes) des Arguments *format* geben die Zahl von Zeichen für jeden Tabulator an. Die Standardzeichenzahl für einen Tabulator ist acht.
DT_TOP	Gibt nach unten ausgerichteten Text an (nur bei Einzelzeile).
DT_VCENTER	Gibt vertikal zentrierten Test an (nur bei Einzelzeile).
DT_WORDBREAK	Gibt Wortumbruch an. Zeilen werden automatisch zwischen Wörtern umgebrochen, wenn ein Wort den Rand des mit *r* angegebenen Rechteckes überschreitet Eine Sequenz Zeilenrücklauf/Zeilenvorschub wird die Zeile ebenfalls umbrechen.

Beachten Sie, daß die Werte DT_CALCRECT, DT_EXTERNALLEADING, DT_INTERNAL, DT_NOCLIP und DT_NOPREFIX nicht zusammen mit dem Wert DT_TABSTOP benutzt werden können.

DrawText verwendet die aktuell ausgewählte Schrift, die Textfarbe und die Hintergrundfarbe dieses DC, um den Text zu zeichnen. Solange das Format DT_NOCLIP nicht benutzt wird, schneidet *DrawText* den Text ab, so daß er nicht außerhalb des gegebenen Rechteckes erscheinen kann. Es wird auf mehrere Zeilen hin formatiert, es sei denn, das Format DT_SINGLELINE ist vorgegeben.

Wenn die ausgewählte Schrift zu groß für das angegebene Rechteck ist, wird *DrawText* nicht versuchen, eine kleinere Schrift einzusetzen.

Bei Erfolg gibt *DrawText* die Höhe des Textes zurück. Andernfalls gibt die Funktion 0 zurück.

Siehe auch: *TDC::GrayString*, *TDC::TabbedTextOut*, *TDC::TextOut*, Klasse *TRect*

Ellipse

```
bool Ellipse(int x1, int y1, int x2, int y2);
bool Ellipse(const TPoint& p1, const TPoint& p2);
bool Ellipse(const TPoint& point, const TSize& size);
bool Ellipse(const TRect& rect);
```

Zeichnet und füllt eine Ellipse auf diesem DC mit Hilfe der aktuell ausgewählten Stift- und Pinselobjekte. Der Mittelpunkt der Ellipse ist zugleich der Mittelpunkt des begrenzenden Rechtecks, das entweder durch (*x1*, *y1*)/(*x2*, *y2*) oder durch das Argument *rect* angegeben ist. *Ellipse* gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück. Die aktuelle Position wird bei diesem Aufruf weder benutzt noch verändert.

Siehe auch: *TDC::Arc*, Klasse *TPoint*, Klasse *TRect*, Klasse *TSize*

EndPath

```
bool EndPath();
```

Nur 32-Bit. Schließt die Pfadsammlung und wählt den Pfad aus, den sie in diesem DC definiert. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::BeginPath*, *TDC::CloseFigure*

EnumFontFamilies

```
int EnumFontFamilies(const char far* family, FONTENUMPROC proc,
                     void* data) const;
```

Zählt die für diesen DC verfügbaren Schriften in der durch *family* gegebenen Schriftfamilie auf. Die gegebenen anwendungsdefinierte Callback-Funktion *proc* (mit *::MakeProcInstance* erzeugt) wird für jede Schrift in der Familie aufgerufen oder bis *proc* 0 zurückgibt. Sie ist wie folgt definiert:

```
typedef int (CALLBACK* FONTENUMPROC)(CONST LOGFONT*, CONST TEXIMETRIC*,
DWORD, LPARAM);
```

data erlaubt Ihnen die Übergabe sowohl von anwendungsspezifischen Daten wie auch von Schriftdaten an *proc*. Bei Erfolg gibt der Aufruf den letzten Rückgabewert von *proc* zurück.

Siehe auch: *TDC::EnumFonts*, Struktur *LOGFONT*, Struktur *TEXTMETRIC*

EnumFonts

```
int EnumFonts(const char far* faceName, OLDFONTENUMPROC callback,
                    void far* data) const;
```

Zählt die auf diesem DC verfügbaren Schriften für das gegebene *faceName* auf. Die Daten des Schrifttyps, LOGFONT, und von TEXTMETRIC werden für jede verfügbare Schrift abgefragt und an die benutzetdefinierte *callback*-Funktion zusammen mit allen optionalen Daten zurückgegeben, die vom Benutzer im Puffer *data* plaziert werden. Die Funktion *callback* kann die Daten auf jede gewünschte Art verarbeiten. Die Aufzählung fährt fort, bis keine Schriften mehr vorliegen oder bis *callback* 0 zurückgibt. Wenn *faceName* 0 ist, wählt *EnumFonts* zufällig aus und zählt eine Schrift für jede verfügbare Schriftart auf. *EnumFonts* gibt den letzten, von *callback()* zurückgegebenen Wert zurück. Beachten Sie, daß OLDFONTENUMPROC nur für Win32 als FONTENUMPROC definiert ist. FONTENUMPROC ist ein Zeiger auf eine benutzerdefinierte Funktion. Sie hat den folgenden Prototyp:

```
int CALLBACK EnumFontsProc(LOGFONT *lplf, TEXTMETRIC *lptm, uint32
dwType,
                           LPARAM lpData);
```

Dabei gibt *dwType* dem Schrifttyp an: DEVICE_FONTTYPE, RASTER_FONTTYPE oder TRUETYPE_FONTTYPE.

Siehe auch: *TDC::EnumFontFamilies*, *Struktur LOGFONT*, *Struktur TEXTMETRIC*

EnumMetaFile

```
int EnumMetaFile(const TMetaFilePict& metaFile, MFENUMPROC
                     callback, void* data) const;
```

Zählt die GDI-Aufrufe (GDI = Graphics Device Interface, Grafische Geräteschnittstelle) innerhalb des gegebenen *metaFile* auf. Jeder solche Aufruf wird wieder angefordert und zusammen mit irgendwelchen Client-Daten von *data* an die gegebene *callback*-Funktion übergeben. Dies erfolgt solange, bis alle Aufrufe verarbeitet wurden oder bis eine Callback-Funktion 0 zurückgibt. *MFENUMPROC* ist wie folgt definiert:

```
typedef int(CALLBACK* MFENUMPROC)(HDC, HANDLETABLE FAR*, METARECORD FAR*,
int, LPARAM);
```

Siehe auch: *TDC::PlayMetaFile,* Struktur *METARECORD*

EnumObjects

```
int EnumObjects(uint objectType, GOBJENUMPROC proc, void far* data) const;
```

Zählt die für diesen DC verfügbaren Stift- oder Pinselobjekte auf. Der Parameter *objectType* kann entweder OBJ_BRUSH oder OBJ_PEN sein. Für jeden gefundenen Pinsel oder Stift wird proc, eine benutzerdefinierte *Callback*-Funktion, aufgerufen, bis keine weiteren Objekte gefunden werden oder die Callback-Funktion 0 zurückgibt. *proc* ist wie folgt definiert:

```
typedef int (CALLBACK* GOBJENUMPROC)(LPVOID, LPARAM);
```

Der Parameter *data* gibt einen anwendungsspezifischen Wert an, der an *proc* übergeben wird.

ExcludeClipRect

```
int ExcludeClipRect(const TRect& rect);
```

Erzeugt einen neuen Clipping-Bereich für diesen DC. Dieser neue Bereich besteht aus dem aktuellen Clipping-Bereich minus dem gegebenen Rechteck *rect*. Der Rückgabewert gibt den Typ des neuen Clipping-Bereichs wie folgt an:

Bereich	Bedeutung
COMPLEXREGION	Clipping-Bereich hat überlappende Rahmen.
ERROR	Ungültiger DC.
NULLREGION	Clipping-Bereich ist leer.
SIMPLEREGION	Clipping-Bereich hat keine überlappende Rahmen.

Siehe auch: *TDC::GetClipBox, Klasse TRect*

ExcludeUpdateRgn

```
int ExcludeUpdateRgn(HWND wnd);
```

Verhindert das Zeichnen innerhalb ungültiger Bereiche eines Fensters, indem ein aktualisierter Bereich des Fensters von diesem DC von dessen Clipping-Bereich ausgeschlossen wird. Der Rückgabewert zeigt den resultierenden Typ des Clipping-Bereichs wie folgt an:

Bereich	Bedeutung
COMPLEXREGION	Clipping-Bereich hat überlappende Rahmen.
ERROR	Ungültiger DC.
NULLREGION	Clipping-Bereich ist leer.
SIMPLEREGION	Clipping-Bereich hat keine überlappende Rahmen.

ExtFloodFill

```
bool ExtFloodFill(const TPoint& point, TColor color,
                  uint16 fillType);
```

Füllt einen Bereich auf diesem DC, der bei *point* startet und das aktuell ausgewählte Pinselobjekt benutzt. Das Argument *color* gibt die Farbe der Umgrenzung oder des zu füllenden Bereichs an. Das Argument *fillType* gibt den Fülltyp wie folgt an:

FLOODFILLBORDER	Der Füllbereich ist durch die gegebene Farbe *color* begrenzt. Dieser Stil fällt mit der in *FloodFill()* verwendeten Füllmethode zusammen.

FLOODFILLSURFACE	Der Füllbereich ist durch die gegebene Farbe *color* definiert. Das Füllen wird nach außen in alle Richtungen fortgesetzt, solange diese Farbe angetroffen wird. Verwenden Sie diesen Stil, wenn Sie Bereiche mit mehrfarbigen Rändern füllen.

Nicht jedes Gerät unterstützt *ExtFloodFill*. Die Anwendungen sollten daher zuerst mit *TDC::GetDeviceCaps* vortesten.

ExtFloodFill gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::FloodFill*, *TDC::GetDeviceCaps*, Klasse *TColor*, Klasse *TPoint*

ExtTextOut

```
virtual bool ExtTextOut(int x, int y, uint16 options,
                        const TRect* r, const char far* string,
                        int count, const int far* dx = 0);

bool ExtTextOut(const TPoint& p, uint16 options, const TRect* r,
                const char far* string, int count,
                const int far* dx = 0);
```

Zeichnet bis zu *count* Zeichen des gegebenen nullterminierten Strings *string* in der aktuellen Schrift auf diesen DC. Wenn *count* -1 ist, wird der ganze String geschrieben.

Ein optionales Rechteck *r* kann für das Clipping oder das durchsichtige Zeichnen (opaquing) oder für beides verwendet werden. Dies wird durch den Wert *options* festgelegt. Wenn *options* auf ETO_CLIPPED gesetzt ist, wird das Rechteck für das Clipping des gezeichneten Textes benutzt. Wenn *options* auf ETO_OPAQUE gesetzt ist, wird die aktuelle Hintergrundfarbe zum Füllen des Rechteckes benutzt. Beide Optionen können benutzt werden, wenn ETO_CLIPPED mit ETO_OPAQUE durch »Oder« verbunden ist.

Die Argumente (*x*, *y*) oder *p* geben die logischen Koordinaten des Referenzpunktes an, der für die Ausrichtung des ersten Zeichens benutzt wird. Der aktuelle Textausrichtungsmodus kann mit *TDC::GetTextAlign* abgefragt und mit *TDC::SetTextAlign* geändert werden. Standardmäßig wird die aktuelle Position durch den Aufruf von *ExtTextOut* weder benutzt noch verändert. Wenn der Ausrichtungsmodus allerdings auf TA_UPDATECP gesetzt ist, ignoriert *ExtTextOut* die Argumente für den Referenzpunkt und verwendet oder aktualisiert die aktuelle Position als den Referenzpunkt.

Das Argument *dx* ist ein optionales Array mit Werten, die die Distanzen zwischen den Ursprüngen (linken oberen Ecken) von benachbarten Zeichenzellen festlegen. So repräsentiert *dx[i]* zum Beispiel die Zahl von logischen Einheiten, die die Ursprünge der Zeichenzellen *i* und *i*+1 angeben. Wenn *dx* 0 ist, verwendet *ExtTextOut* den Standardzeichenabstand.

ExtTextOut gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::TextOut*, *TDC::GetTextAlign*, *TDC::TabbedTextOut*, *TPoint*, *TRect*

FillPath

```
bool FillPath();
```

Nur 32-Bit. Schließt alle offenen Figuren im aktuellen Pfad dieses DC und füllt das Innere des Pfades mit Hilfe das aktuellen Pinsels und Polygonfüllmodus. Nach dem Füllen des Innenteils entfernt *FillPath* den Pfad vom DC.

FillPath gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::BeginPath*, *TDC::CloseFigure*, *TDC::StrokePath*, *TDC::StrokeAndFillPath*, *TDC::SetPolyFillMode*

FillRect

```
bool FillRect(int x1, int x2, int y1, int y2, const TBrush& brush);
```

```
bool FillRect(const TRect& rect, const TBrush& brush);
```

Füllt das gegebene Rechteck auf diesem DC mit Hilfe des angegebenen Pinsels. Die Füllung umfaßt den linken und oberen Rahmen, sie schließt allerdings den rechten und unteren Rahmen aus. *FillRect* gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TBrush*, *TRect*

FillRgn

```
bool FillRgn(const TRegion& region, const TBrush& brush);
```

Füllt den gegebenen Bereich *region* auf diesem DC mit Hilfe des angegeben Pinsels *brush*. *FillRgn* gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::InvertRgn*, *TDC::PaintRgn*, Klasse *TBrush*, Klasse *TRegion*

FlattenPath

```
bool FlattenPath();
```

Nur 32-Bit. Transformiert alle Kurven im aktuell ausgewählten Pfad von diesem DC. Alle solchen Kurven werden in Sequenzen linearer Segmente geändert. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::WidenPath*, *TDC::BeginPath*

FloodFill

```
bool FloodFill(const TPoint& point, TColor color);
```

Füllt einen Bereich auf diesem DC, der bei *point* startet, mit Hilfe des aktuell ausgewählten Pinselobjektes. Das Argument *color* gibt die Farbe der Umgrenzung oder des zu füllenden Bereichs an. Gibt TRUE zurück,

wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück. *FloodFill* ist in der WIN32-API aus Gründen der Kompatibilität zu früheren APIs beibehalten worden. Neue WIN32-Anwendungen sollten *TDC::ExtFloodFill()* verwenden.

Siehe auch: *TDC::ExtFloodFill*, Klasse *TColor*, Klasse *TPoint*

FrameRect

```
bool FrameRect(int x1, int x2, int y1, int y2,
                    const TBrush& brush);

bool FrameRect(const TRect& rect, const TBrush& brush);
```

Zeichnet einen Rahmen auf diesem DC um das gegebene Rechteck herum mit Hilfe des gegebenen Pinsels. Die Höhe und Breite des Rahmens ist eine logische Einheit. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: Klasse *TBrush*, Klasse *TRect*

FrameRgn

```
bool FrameRgn(const TRegion& region, const TBrush& brush,
                    const TPoint& p);
```

Zeichnet einen Rahmen auf diesem DC um den gegebenen Bereich *region* herum mit Hilfe des gegebenen Pinsels *brush*. Die Breite und Höhe des Rahmens ist durch das Argument *p* angegeben. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: Klasse *TBrush*, Klasse *TRegion*

GetAspectRatioFilter

```
bool GetAspectRatioFilter(TSize& size) const;
```

Fragt die Einstellung des aktuellen XY-Verhältnisfilters für dieses DC ab.

Siehe auch: *TSize*

GetBkColor

```
TColor GetBkColor() const;
```

Gibt die aktuelle Hintergrundfarbe dieses DC wieder.

Siehe auch: *TDC::SetBkColor*, Klasse *TColor*

GetBkMode

```
int GetBkMode() const;
```

Gibt den Hintergrundmodus für dieses DC zurück. Dies ist entweder OPAQUE oder TRANSPARENT.

Siehe auch: *TDC::SetBkMode*

GetBoundsRect

```
bool GetBoundsRect(TRect& bounds, uint16 flags) const;
```

Gibt in *bounds* das aktuell beibehaltene begrenzende Rechteck dieses DC oder der Windows-Verwaltung an, abhängig vom Wert *flags*. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Das Argument *flags* kann DCB_RESET oder DCB_WINDOWMGR oder beides sein. Der Wert *flags* hat folgende Bedeutung:

DCB_RESET	Das begrenzende Rechteck wird gelöscht, nachdem es in *bounds* gesetzt wurde.
DCB_WINDOWMGR	Gibt das aktuelle Grenzrechteck von Windows und nicht das von diesem DC wieder.

Es werden zwei begrenzende Rechtecke beibehalten, eines für Windows und eines für die Anwendung. *GetBoundsRect* gibt für die Windows-Grenzen die Bildschirmkoordinaten und für die Anwendungsgrenzen die logischen Einheiten zurück. Die für Windows beibehaltenen Grenzen können durch eine Anwendung zwar erfragt, jedoch nicht geändert werden. Die Anwendung kann aber die beibehaltenen Grenzen des DC sowohl abfragen als auch ändern.

Siehe auch: *TDC::SetBoundsRect*, Klasse *TRect*

GetBrushOrg

```
bool GetBrushOrg(TPoint& point) const;
```

Plaziert in *point* den aktuellen Pinselursprung dieses DC. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: Klasse *TPoint*

GetCharABCWidths

```
bool GetCharABCWidths(uint firstChar, uint lastChar, ABC* abc);
```

Ermittelt die Breite aufeinanderfolgender Zeichen im Bereich von *firstChar* bis *lastChar* von der aktuellen TrueType-Schrift dieses DC. Die Breiten werden in dem Array *abc* der *ABC*-Strukturen wiedergegeben. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *::GetCharABCWidths*, Struktur *ABC*

GetCharWidth

```
bool GetCharWidth(uint firstChar, uint lastChar, int* buffer);
```

Ermittelt die Breite in einer aufeinanderfolgenden Zeichenreihe in logischen Einheiten in der aktuellen Schrift dieses DC. Die Reihe ist durch den Bereich von *firstChar* bis *lastChar* (einschließlich) angegeben und die Breiten werden in den gegebenen *buffer* kopiert. Wenn ein Zeichen in dem Bereich nicht in der aktuellen Schrift repräsentiert ist, wird die Breite des Standardzeichens zugewiesen. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::GetCharABCWidths*

GetClipBox

```
int GetClipBox(TRect& rect) const;

TRect GetClipBox() const;
```

Plaziert das aktuelle Clipping-Feld dieses DC in *rect*. Das Clipping-Feld ist definiert als das kleinste Rechteck, das die aktuelle Clipping-Grenze

umgibt. Der Rückgabewert zeigt den Typ des Clipping-Bereichs wie folgt an:

Bereich	Bedeutung
COMPLEXREGION	Clipping-Bereich hat überlappende Rahmen.
ERROR	Ungültiger DC.
NULLREGION	Clipping-Bereich ist leer.
SIMPLEREGION	Clipping-Bereich hat keine überlappende Rahmen.

Siehe auch: *TDC::ExcludeClipRect*, Klasse *TRect*

GetClipRgn

```
bool GetClipRgn(TRegion& region) const;
```

Ermittelt den aktuellen Clipping-Bereich dieses DC und plaziert eine *Kopie* davon in das Argument *region*. Sie können diese Kopie ändern, ohne den aktuellen Clipping-Bereich zu beeinflussen. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: Klasse *TRegion*

GetCurrentObject

```
HANDLE GetCurrentObject(uint objectType) const;
```

Gibt ein Handle auf das aktuell ausgewählte Objekt des gegebenen Typs *objectType*, das mit diesem DC assoziiert ist, zurück. Die Funktion gibt 0 zurück, wenn der Aufruf fehlschlägt. *objectType* kann OBJ_PEN, OBJ_BRUSH, OBJ_PAL, OBJ_FONT oder OBJ_BITMAP sein.

GetCurrentPosition

```
bool GetCurrentPosition(TPoint& point) const;
```

Gibt in *point* die logischen Koordinaten der aktuellen Position dieses DC wieder. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: Klasse *TPoint*

GetDCOrg

```
bool GetDCOrg(TPoint& point) const;
```

Erhält den endgültigen Verschiebungsursprung für diesen Gerätekontext und plaziert den Wert in *point*. Dieser Wert gibt den Offset an, der für das Umsetzen der Gerätekoordinaten in Client-Koordinaten für Punkte in einem Anwendungsfenster benutzt wird. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: Klasse *TPoint*

GetDeviceCaps

```
virtual int GetDeviceCaps(int index) const;
```

Wenn die Funktion *GetDeviceCaps* unter WIN3.1 oder unter späteren Versionen verwendet wird, gibt sie Informationen über Fähigkeiten dieses DC zurück. Das Argument *index* gibt den Typ der geforderten Information an.

GetDIBits

```
bool GetDIBits(const TBitmap& bitmap, uint16 startScan,
                     uint16 numScans, void HUGE* bits,
                     const BITMAPINFO far& info, uint16 usage);

bool GetDIBits(const TBitmap& bitmap, TDib& dib);
```

Die erste Version ermittelt einige oder alle Bits der gegebenen Bitmap *bitmap* auf diesem DC und kopiert sie in den Puffer *bits*. Dabei wird ein DIB-Format (DIB = Device-Independent Bitmap, geräteunabhängige Bitmap) verwendet, das durch das BITMAPINFO-Argument *info* festgelegt wird. *numScan* Zeilen der Bitmap, die bei Zeile *startScan* starten, werden ermittelt. Das Argument *usage* bestimmt das Format des BITMAPINFO-Strukturelementes *bmiColors* entsprechend der folgenden Tabelle:

Wert	Bedeutung
DIB_PAL_COLORS	Die Farbtabelle ist ein Array mit 16-Bit-Indizes auf die aktuelle logische Palette.
DIB_RGB_COLORS	Die Farbtabelle enthält echte RGB-Werte.
DIB_PAL_INDICES	Es gibt keine Farbtabelle für diese Bitmap. Die DIB-Bits bestehen aus Indizes auf die Systempalette. Es findet keine Farbübersetzung statt. Nur der BITMAPINFOHEADER-Teil von BITMAPINO wird gefüllt.

Bei der zweiten Version von *GetDIBits* werden die Bits von *bitmap* ermittelt und in das Datenelement *dib.Bits* des gegebenen Argumentes *TDib* plaziert. Das BITMAPINFO-Argument wird von *dib.info* geliefert.

GetDIBits gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::SetDIBits*, Klasse *TBitmap*, Klasse *TDib*, Struktur *BITMAPINFO*

GetFontData

```
uint32 GetFontData(uint32 table, uint32 offset, void* buffer,
                   long data);
```

Ermittelt schriftmetrische Informationen von einer skalierbaren TrueType-Schriftdatei (angegeben durch *table* und in der Tabelle startend bei *offset*) und plaziert die Information in den gegeben Puffer *buffer*. *data* gibt die Größe der zu ermittelnden Daten in Bytes an. Wenn der Aufruf erfolgreich ist, gibt die Funktion die Zahl der in *buffer* gesetzten Bytes zurück. Andernfalls gibt sie -1 zurück.

GetGlyphOutline

```
uint32 GetGlyphOutline (uint chr, uint format, GLYPHMETRICS far& gm,
uint32 buffSize, voidfar* buffer, const MAT2 far& mat2);
```

Ermittelt TrueType-Metrik und andere Daten für das gegebene Zeichen *chr* auf diesem DC und plaziert sie in *gm* und *buffer*. Das Argument *format* gibt das Format der ermittelten Daten wie folgt an:

Wert	Bedeutung
1	Ermittelt die Bitmap des Zeichenkörpers.
2	Ermittelt die Kurvendatenpunkte im ursprünglichen Format der Rasterung und verwendet die Gestaltungseinheiten der Schrift. Mit diesem Wert von *format* wird das Transformationsargument *mat2* ignoriert.

Das Argument *gm* gibt die GLYPHMETRICS-Struktur an, die die Plazierung des Zeichenkörpers in der Zeichenzelle beschreibt. *buffSize* legt die Größe von *buffer* fest, der Daten über das Umrißzeichen empfängt. Wenn entweder *buffSize* oder *buffer* 0 sind, gibt *GetGlyphOutline* die erforderliche Puffergröße zurück. Anwendungen können im Bitmap-Format (*format* = 1) ermittelte Zeichen drehen, indem eine 2 x 2-Transformationsmatrix über das Argument *mat2* angegeben wird.

GetGlyphOutline gibt eine positive Zahl zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion GDI_ERROR zurück.

Siehe auch Struktur *GLYPHMETRICS*, *TDC::GetOutlineTextMetrics*

GetKerningPairs

```
int GetKerningPairs(int pairs, KERNINGPAIR far* krnPair);
```

Ermittelt bis zu *pairs* Kerning-Paare für die aktuelle Schrift dieses DC und kopiert sie in das Array *krnPair* mit KERNINGPAIR-Strukturen. Bei Erfolg gibt die Funktion die aktuelle Zahl ermittelter Paare zurück. Wenn die Schrift mehr als *pairs* Kerning-Paare besitzt, schlägt der Aufruf fehl und liefert 0 zurück. Das Array *krnPair* muß mindestens *pairs* KERNINGPAIRS-Strukturen zulassen. Wenn *krnPair* auf 0 gesetzt ist, gibt *GetKerningPairs* die Gesamtzahl von Kerning-Paaren in der aktuellen Schrift zurück.

Siehe auch: Struktur *KERNINGPAIR*

GetMapMode

```
int GetMapMode() const;
```

Bei Erfolg gibt *GetMapMode* den aktuellen Fensterabbildungsmodus dieses DC zurück. Andernfalls gibt die Funktion 0 zurück. Der Abbildungsmodus definiert, wie logische Koordinaten in Gerätekoordinaten abgebildet werden. Er kontrolliert ferner die Orientierung der x- und y-Achsen des Gerätes. Die Moduswerte sind in der folgenden Tabelle aufgelistet:

Wert	Bedeutung
MM_ANISOTROPIC	Logische Einheiten werden auf beliebige Einheiten mit beliebig skalierten Achsen abgebildet. Die Funktionen *SetWindowExtEx* und *SetViewportExtEx* müssen für die Angabe der gewünschten Einheiten, der Orientierung und der Skalierung verwendet werden.

MM_HIENGLISH	Jede logische Einheit wird auf 0.001 Zoll (1 Zoll = 2.54 cm) abgebildet. Positives *x* geht nach rechts und positives *y* geht nach oben.
MM_HIMETRIC	Jede logische Einheit wird auf 0.01 Millimeter abgebildet. Positives *x* geht nach rechts und positives *y* geht nach oben.
MM_ISOTROPIC	Logische Einheiten werden auf beliebige Einheiten mit gleichartig skalierten Achsen abgebildet. Das heißt, eine Einheit auf der x-Achse ist gleich einer Einheit auf der y-Achse. Die Funktionen *SetWindowExtEx* und *SetViewportExtEx* müssen für die Angabe der gewünschten Einheiten und der Orientierung der Achsen verwendet werden. GDI nimmt je nach Erfordernis Anpassungen vor, um sicherzustellen, daß die x- und y-Einheiten in derselben Größe bleiben (wenn Sie zum Beispiel die Fensterausdehnung neu bestimmen, wird der Viewport angepaßt, um die Einheiten isotrop zu halten).
MM_LOENGLISH	Jede logische Einheit wird auf 0.01 Zoll (1 Zoll = 2.54 cm) abgebildet. Positives *x* geht nach rechts und positives *y* geht nach oben.
MM_LOMETRIC	Jede logische Einheit wird auf 0.1 Millimeter abgebildet. Positives *x* geht nach rechts und positives *y* geht nach oben.
MM_TEXT	Jede logische Einheit wird auf einen einzigen Gerätepixel abgebildet. Positives *x* geht nach rechts und positives *y* geht nach unten.
MM_TWIPS	Jede logische Einheit wird auf ein Zwanzigstel des Punktmaßes eines Druckers (1/1440 Zoll, 1 Zoll = 2.54 cm) abgebildet. Positives *x* geht nach rechts und positives *y* geht nach oben.

Siehe auch: *TDC::SetMapMode*

GetNearestColor

```
TColor GetNearestColor(TColor Color) const;
```

Gibt für die aktuelle Palette dieses DC die dem gegebenen Argument *Color* nächstgelegene Farbe zurück.

Siehe auch: Klasse *TColor*

GetOutlineTextMetrics

```
uint32 GetOutlineTextMetrics(uint data, OUTLINETEXTMETRIC far& otm);
uint16 GetOutlineTextMetrics(uint data, OUTLINETEXTMETRIC far& otm);
```

Ermittelt metrische Information für TrueType-Schriften auf diesem DC und kopiert sie in das gegebene Array mit OUTLINETEXTMETRIC-Strukturen *otm*. Diese Struktur enthält ein TEXTMETRIC, mehrere andere metrische Elemente sowie vier String-Zeigerelemente für die Namen der Schriftfamilie, der Schrift, des Stils und für den vollständigen Schriftnamen. Da für diese Strings mit variabler Länge zusätzlich zu den Metrikdaten der Schrift Speicherplatz reserviert werden muß, müssen Sie die Gesamtzahl der für die ermittelten Daten erforderlichen Bytes (über das Argument *data*) übergeben. Wenn *GetOutlineTextMetrics* mit *otm* = 0 aufgerufen wird, gibt die Funktion die erforderliche Gesamtgröße des Puffers zurück. Sie können diesen Wert dann an *data* in nachfolgenden Aufrufen zuweisen.

Gibt einen Wert ungleich Null zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion Null zurück.

Siehe auch:Struktur *OUTLINETEXTMETRICS*,Struktur *TEXTMETRIC*, *TCD::GetTextMetrics*

GetPixel

```
TColor GetPixel(int x, int y) const;
TColor GetPixel(const TPoint& point) const;
```

Gibt die Farbe des Pixel an der gegebenen Position zurück.

Siehe auch: *TDC::SetPixel*, *Klasse TPoint*

GetPolyFillMode

```
int GetPolyFillMode() const;
```

Gibt den aktuellen Polygon-Füllmodus für diesen DC zurück, entweder ALTERNATE oder WINDING.

Siehe auch: *TDC::SetPolyFillMode*

GetROP2()

```
int GetROP2() const;
```

Gibt den aktuellen Zeichenmodus (Rasteroperation) dieses DC zurück.

Siehe auch: *TDC::SetROP2*

GetStretchBltMode

```
int GetStretchBltMode() const;
```

Gibt den aktuellen Streckmodus für diesen DC zurück: BLACKONWHITE, COLORONCOLOR oder WHITEONBLACK. Der Streckmodus gibt an, wie Bitmaps durch die Funktion *StretchBlt* gestaucht oder gestreckt werden.

Siehe auch: *TDC::SetStretchBltMode*, *TDC::StretchBlt*

GetSystemPaletteEntries

```
uint GetSystemPaletteEntries(int start, int num, PALETTEENTRY
                                        far* entries) const;
```

Ermittelt, bei *start* beginnend, einen Bereich von bis zu *num* Paletteneinträgen aus der Systempalette und legt sie im Array *entries* mit PALETTEENTRY-Strukturen ab. Gibt die aktuelle Zahl von transferierten Einträgen zurück.

Siehe auch: Strukur *PALETTEENTRY*

GetSystemPaletteUse

```
uint GetSystemPaletteUse() const;
```

Bestimmt, ob dieser DC Zugriff auf die gesamte Systempalette besitzt. Gibt SYSPAL_NOSTATIC oder SYSPAL_STATIC zurück.

Siehe auch: *TDC::SetSystemPaletteUse*

GetTabbedTextExtent

```
bool GetTabbedTextExtent(const char far* string, int stringLen,
                                int numPositions, const int* positions,
                                TSize& size) const;

TSize GetTabbedTextExtent(const char far* string, int stringLen,
                                int numPositions, const int* positions)
                                const;
```

Berechnet in logischen Einheiten die Ausdehnung (Breite und Höhe) der Textzeile, die aus *stringLen* Zeichen des nullterminierten Strings *string* besteht. Die Ausdehnung wird von den metrischen Informationen der aktuellen Schrift oder dieses DC berechnet. Dabei wird allerdings der aktuelle Clipping-Bereich ignoriert. In der ersten Version von *GetTabbedTextExtent* wird die Ausdehnung in *size* zurückgegeben, in der zweiten Version wird die Ausdehnung als *TSize*-Objekt zurückgegeben. Die Breite ist *size.x* und die Höhe ist *size.y*.

Die Berechnung der Breite beinhaltet die Zwischenräume, die mit jedem Tabulator-Code im String vorgegeben sind. Solche Tabulator-Codes werden mit Hilfe der Argumente *numPositions* und *positions* interpretiert. Das Array *positions* gibt *numPositions* Spaltenformate, in Geräteeinheiten gemessen, an. Die Spaltenformate müssen im Array streng anwachsende Werte haben. Wenn sowohl *numPositions* als auch *positions* 0 sind, werden Tabulatoren auf das Achtfache einer durchschnittlichen Zeichenbreite expandiert. Wenn *numPositions* 1 ist, werden die Spaltenformate *positions[0]* Einheiten voneinander getrennt.

Wenn Kerning angewendet wird, kann die Summe der Ausdehnungen der Zeichen in einem String von der Ausdehnung des gesamten Strings abweichen.

Siehe auch: *TDC::TabbedTextOut*, *TDC::GetTextExtent*, Klasse *TSize*

GetTextAlign

```
uint GetTextAlign() const;
```

Bei Erfolg gibt *GetTextAlign* die aktuellen Flags dieses DC für die Textausrichtung zurück. Andernfalls gibt die Funktion den Wert GDI_ERROR zurück. Die Textausrichtungs-Flags legen fest, wie *TDC::TextOut()* und *TDC::ExtTextOut* Text-Strings im Verhältnis zur Bildschirmposition des ersten Zeichens ausrichten. *GetTextAlign* gibt bestimmte Flag-Kombinationen zurück, die in der folgenden Tabelle aufgelistet sind:

Wert	Bedeutung
TA_BASELINE	Der Referenzpunkt liegt auf der Basislinie des Textes.
TA_BOTTOM	Der Referenzpunkt liegt auf der Bodenkante des begrenzenden Rechtecks.
TA_TOP	Der Referenzpunkt liegt auf der oberen Kante des begrenzenden Rechtecks.
TA_CENTER	Der Referenzpunkt liegt zur Mitte des begrenzenden Rechtecks horizontal ausgerichtet.
TA_LEFT	Der Referenzpunkt liegt auf der linken Kante des begrenzenden Rechtecks.
TA_RIGHT	Der Referenzpunkt liegt auf der rechten Kante des begrenzenden Rechtecks.
TA_NOUPDATECP	Die aktuelle Position wird nicht nach jedem Aufruf einer Textausgabe aktualisiert.
TA_UPDATECP	Die aktuelle Position wird nach jedem Aufruf einer Textausgabe aktualisiert. Wenn die aktuelle Schrift eine vertikale Standardbasislinie besitzt

(wie bei Kanji), ersetzen die folgenden Werte TA_BASELINE und TA_CENTER:

VTA_BASELINE	Der Referenzpunkt liegt auf der Basislinie des Textes.
VTA_CENTER	Der Referenzpunkt liegt zur Mitte des begrenzenden Rechtecks vertikal ausgerichtet.

Die Flags zur Textausrichtung sind nicht zwingend Einzel-Bit-Flags und können gleich 0 sein. Die Flags müssen in den folgenden Gruppen zusammengehöriger Flags geprüft werden:

- TA_LEFT, TA_RIGHT und TA_CENTER
- TA_BOTTOM, TA_TOP und TA_BASELINE
- TA_NOUPDATECP und TA_UPDATECP

Wenn die aktuelle Schrift eine vertikale Standardbasislinie besitzt (wie bei Kanji), sind die zusammengehörigen Flags wie folgt gruppiert:

- TA_LEFT, TA_RIGHT und VTA_BASELINE
- TA_BOTTOM, TA_TOP und VTA_CENTER
- TA_NOUPDATECP und TA_UPDATECP

Um sicherzustellen, daß ein bestimmtes Flag im Rückgabewert dieser Funktion gesetzt ist, muß die Anwendung die folgenden Schritte ausführen:

1. Sie muß den bitweisen Oder-Operator auf das Flag und dessen zugehörigen Flags anwenden.
2. Sie muß den bitweisen Und-Operator auf das Ergebnis und den Rückgabewert anwenden.
3. Sie muß auf Gleichheit dieses Ergebnisses und des Flags testen.

Das folgende Beispiel zeigt eine Methode zur Bestimmung, welches Flag für die horizontale Ausrichtung gesetzt ist.

```
switch ((TA_LEFT | TA_RIGHT | TA_CENTER) & dc.GetTextAlign()) {
   case TA_LEFT:
      ...
   case TA_RIGHT:
      ...
   case TA_CENTER:
      ...
}
```

Siehe auch: *TDC::SetTextAlign*, *TDC::TextOut*, *TDC::ExtTextOut*

GetTextCharacterExtra `int GetTextCharacterExtra() const;`

Bei Erfolg gibt die Funktion den Zeichenabstand für diesen DC in logischen Einheiten zurück. Andernfalls gibt sie INVALID_WIDTH zurück.

Siehe auch Elementfunktion *TDC::SetTextCharacterExtra*

GetTextColor

```
TColor GetTextColor() const;
```

Gibt die aktuelle Textfarbe dieses DC zurück. Die Textfarbe legt die von *TDC::TextOut()* und *TDC::ExtTextOut()* angezeigte Farbe fest.

Siehe auch: *TDC::SetTextColor*, *TDC::TextOut*, *TDC::ExtTextOut*, *TColor*

GetTextExtent

```
bool GetTextExtent(const char far* string, int stringLen,
                   TSize& size);

TSize GetTextExtent(const char far* string, int stringLen);
```

Berechnet in logischen Einheiten die Ausdehnung (Breite und Höhe) der Textzeile, die aus *stringLen* Zeichen des nullterminierten Strings *string* besteht. Die Ausdehnung wird von den metrischen Informationen der aktuellen Schrift oder dieses DC berechnet. Dabei wird allerdings der aktuelle Clipping-Bereich ignoriert. In der ersten Version von *GetTextExtent* wird die Ausdehnung in *size* zurückgegeben, in der zweiten Version wird die Ausdehnung als *TSize*-Objekt zurückgegeben. Die Breite ist *size.x* und die Höhe ist *size.y*.

Wenn Kerning angewendet wird, kann die Summe der Ausdehnungen der Zeichen in einem String von der Ausdehnung des gesamten Strings abweichen.

GetTextExtent wird unter WIN32 nicht unterstützt. WIN32-Anwendungen sollten *TDC::GetTextExtentPoint* verwenden.

Siehe auch: Klasse *TSize*

GetTextFace

```
int GetTextFace(int count, char far* facename) const;
```

Ermittelt den Schriftartnamen der aktuellen Schrift auf diesem DC. Bis zu *count* Zeichen dieses Namens werden nach *facename* kopiert. Bei Erfolg gibt *GetTextFace* die Zahl der aktuell kopierten Zeichen zurück. Andernfalls gibt die Funktion 0 zurück.

Siehe auch: *TDC::GetTextAlign*, *TDC::GetTextMetrics*

GetTextMetrics

```
bool GetTextMetrics(TEXTMETRIC far& metrics) const;
```

Füllt die Struktur *metrics* mit metrischen Daten für die aktuelle Schrift auf diesem DC. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: Struktur *TEXTMETRIC*

GetViewportExt

```
bool GetViewportExt(TSize& extent) const;
TSize GetViewportExt() const;
```

Die erste Version ermittelt die x- und y-Ausdehnungen des Viewport dieses DC (in Geräteeinheiten) und plaziert die Werte in *extent*. Diese Version gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die

Funktion FALSE zurück. Die zweite Version gibt nur diese x- und y-Ausdehnungen zurück.

Der Wert *extent* bestimmt den im logischen Koordinatensystem erforderlichen Umfang des Streckens oder Stauchens für eine Anpassung an das Koordinatensystem des Gerätes. *extent* bestimmt ferner die relative Orientierung der beiden Koordinatensysteme.

Siehe auch: *TDC::SetViewportExt*, Klasse *TSize*

GetViewportOrg

```
bool GetViewportOrg(TPoint& point) const;
TPoint GetViewportOrg() const;
```

Die erste Version setzt im Argument *point* die x- und y-Ausdehnungen des Viewport dieses DC (in Geräteeinheiten). Sie gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück. Die zweite Version gibt die x- und y-Ausdehnungen des Viewport dieses DC (in Geräteeinheiten) zurück.

Siehe auch: *TDC::SetViewportOrg*, *TDC::OffsetViewportOrg*, Klasse *TPoint*

GetWindowExt

```
bool GetWindowExt(TSize& extent) const;
TSize GetWindowExt() const;
```

Ermittelt die aktuellen x- und y-Ausdehnungen des Fensters dieses DC (in Geräteeinheiten). Die erste Version plaziert die Werte in *extent*. und gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück. Die zweite Version gibt die aktuellen Ausdehnungswerte zurück. Der Wert *extent* bestimmt den im logischen Koordinatensystem erforderlichen Umfang des Streckens oder Stauchens für eine Anpassung an das Koordinatensystem des Geräts. *extent* bestimmt ferner die relative Orientierung der beiden Koordinatensysteme.

Siehe auch: *TDC::SetWindowExt*, Klasse *TSize*

GetWindowOrg

```
bool GetWindowOrg(TPoint& point) const;

TPoint GetWindowOrg() const;
```

Plaziert in *point* die x- und y-Koordinaten des mit diesem DC assoziierten Fensterursprungs. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::SetWindowOrg*, *TDC::OffsetWindowOrg*, Klasse *TPoint*

GrayString

```
virtual bool GrayString(const TBrush& brush,
                        GRAYSTRINGPROC outputFunc,
                        const char far* string, int count,
                        const TRect& r);
```

Zeichnet in dem gegebenen Rechteck (*r*) bis zu *count* Zeichen grauen Text vom String *string* mit Hilfe des gegebenen Pinsels *brush* und der aktuellen Schrift für diesen DC. Wenn *count* -1 und *string* nullterminiert ist, wird

der ganze String geschrieben. Das Rechteck muß in logischen Einheiten angegeben werden.

Wenn *brush* 0 ist, wird der Text mit demselben Pinsel grau dargestellt, der auch für das Zeichnen von Fenstertext auf diesem DC verwendet wird. Grauer Text wird primär dazu verwendet, nicht anwählbare Kommandos und Menüeinträge anzuzeigen.

GrayString schreibt den ausgewählten Text in eine Speicher-Bitmap, färbt die Bitmap grau und zeigt das Ergebnis schließlich an. Die Graufärbung wird unabhängig vom aktuellen Pinsel und der aktuellen Hintergrundfarbe ausgeführt.

Der Funktionszeiger *outputFunc* kann die Prozedurinstanz einer anwendungsdefinierten Zeichenfunktion angeben und ist wie folgt definiert:

```
typedef BOOLCCALLBACK* GRAYSTRINGPROC)(HDC, LPARAM, int);
```

Wenn *outputFunc* 0 ist, verwendet *GrayString TextOut* und bei *string* wird von einem normalen, nullterminierten String ausgegangen. Wenn *string* durch *TextOut* nicht behandelt werden kann (wenn der String zum Beispiel als eine Bitmap gespeichert ist), müssen Sie über *outputFunc* eine passende Zeichenfunktion bereitstellen.

Wenn das Gerät eine solide Graufarbe bietet, können graue Strings auch ohne die Funktion *GrayString* direkt gezeichnet werden. Rufen Sie *GetSysColor* auf, um den Farbwert zu finden; zum Beispiel *G* von COLOR_GRAYTEXT. Wenn *G* ungleich Null ist (nicht schwarz), können Sie die Textfarbe mit *SetTextColor(G)* setzen und dann jede brauchbare Funktion verwenden.

GrayString gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück. Ein Fehlschlag kann dann auftreten, wenn *TextOut* oder *outputFunc* FALSE zurückgeben oder wenn nicht genügend Speicher für das Erzeugen der Bitmap zur Verfügung steht.

Siehe auch: *TDC::TextOut*, Klasse *TBrush*, Klasse *TRect*

operator HDC()

```
operator HDC() const{return Handle;}
```

Typumwandlungsoperator. Konvertiert einen Zeiger auf den Typ *HDC* (dem Windows-Datentyp, der das Handle auf einen DC repräsentiert).

IntersectClipRect

```
int IntersectClipRect(const TRect& rect);
```

Erzeugt einen neuen Clipping-Bereich für das Fenster dieses DC. Dazu wird der aktuelle Bereich mit den durch *rect* angegebenen Rechteck geschnitten. Der Rückgabewert gibt den Typ des resultierenden Clipping-Bereichs wie folgt an:

Bereich	Bedeutung
COMPLEXREGION	Clipping-Bereich hat überlappende Rahmen.
ERROR	Ungültiger DC.
NULLREGION	Clipping-Bereich ist leer.
SIMPLEREGION	Clipping-Bereich hat keine überlappende Rahmen.

Siehe auch: *TDC::GetClipBox*, Klasse *TRect*

InvertRect

```
bool InvertRect(int x1, int x2, int y1, int y2);
bool InvertRect(const TRect& rect);
```

Invertiert das gegebene Rechteck auf diesem DC. Bei Monochrombildschirmen werden schwarze und weiße Pixel vertauscht. Bei Farbbildschirmen hängt die Invertierung davon ab, wie die Farben für spezielle Bildschirme generiert werden. Eine gerade Anzahl von Aufrufen der Funktion *InvertRect* stellt die Originalfarben wieder her. *InvertRect* gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: Klasse *TRect*

InvertRgn

```
bool InvertRgn(const TRegion& region);
```

Invertiert den gegebenen Bereich *region* auf diesem DC. Bei Monochrombildschirmen werden schwarze und weiße Pixel vertauscht. Bei Farbbildschirmen hängt die Invertierung davon ab, wie die Farben für spezielle Bildschirme generiert werden. Eine gerade Anzahl (n>=2) von Aufrufen der Funktion *InvertRgn* stellt die Originalfarben wieder her. *InvertRgn* gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::PaintRgn*, *TDC::FillRgn*, Klasse *TRegion*

LineTo

```
bool LineTo(int x, int y);
bool LineTo(const TPoint& point);
```

Zeichnet eine Linie auf diesem DC mit Hilfe des aktuellen Stiftobjektes. Die Linie wird von der aktuellen Position bis zum gegebenen Endpunkt (den Endpunkt selbst ausgenommen) gezogen. Dabei ist der Endpunkt durch (*x*, *y*) oder durch *point* angegeben. Wenn der Aufruf erfolgreich ist, gibt *LineTo* TRUE zurück und der aktuelle Punkt wird auf *point* zurückgesetzt. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: Klasse *TPoint*

LPtoDP

```
bool LPtoDP(TPoint* points, int count = 1) const;
```

Konvertiert jeden der *count* Punkte im Array *points* von logischen Punkten in gerätespezifische Punkte. Die Konvertierung hängt vom Abbil-

dungsmodus auf diesem DC und von den Einstellungen der Ursprünge und Ausdehnungen seines Fensters und Viewport ab. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::DPtoLP*, Klasse *TPoint*

MaskBlt

```
bool MaskBlt(const TRect& dst, const TDC& srcDC, const TPoint& src,
             const TBitmap& maskBm, const TPoint& maskPos,
             uint rop=SRCCOPY);
```

Kopiert eine Bitmap vom gegebenen Quell-DC in diesen DC. *MaskBlt* kombiniert die Farbdaten der Quell- und Ziel-Bitmaps mit Hilfe der gegebenen Maske und Rasteroperation. Das Argument *srcDC* gibt den DC an, von dem die Quell-Bitmap kopiert wird. Die Ziel-Bitmap ist durch das Rechteck *dst* gegeben. Die Quell-Bitmap hat dieselbe Breite und Höhe wie *dst*. Das Argument *src* gibt die logischen Koordinaten der linken oberen Ecke der Quell-Bitmap an. Das Argument *maskBm* gibt eine monochrome Masken-Bitmap an. Wenn *maskBm* nicht monochrom ist, tritt ein Fehler auf. Das Argument *maskPos* gibt die Koordinaten der linken oberen Ecke der Maske an. Der Code für die Rasteroperation *rop* gibt an, wie die Quell-Masken- und Ziel-Bitmaps für die neue Ziel-Bitmap kombiniert werden. Die Codes für die Rasteroperation sind wie folgt:

Wert von *rop*	Bedeutung
BLACKNESS	Füllt *dst* mit der Index-0-Farbe der physikalischen Palette (standardmäßig ist dies Schwarz).
DSTINVERT	Invertiert *dst*.
MERGECOPY	Vermischt die Farben der Quelle und der Maske mit einer booleschen UND-Operation.
MERGEPAINT	Vermischt die Farben der invertierten Quelle mit den Farben von *dst* mit einer booleschen ODER-Operation.
NOTSRCCOPY	Kopiert die invertierte Quelle nach *dst*.
NOTSRCERASE	Kombiniert die Farben der Quelle und von *dst* mit einer booleschen ODER-Operation und invertiert das Ergebnis.
PATCOPY	Kopiert die Maske nach *dst*.
PATINVERT	Kombiniert die Farben der Maske mit den Farben von *dst* mit einer booleschen XOR-Operation.
PATPAINT	Kombiniert die Farben der Maske mit den Farben der invertierten Quelle mit einer booleschen ODER-Operation und kombiniert dann das Ergebnis mit den Farben von *dst* mit einer booleschen ODER-Operation.
SRCAND	Kombiniert die Farben der Quelle und von *dst* mit einer booleschen UND-Operation.
SRCCOPY	Kopiert die Quelle direkt nach *dst*.
SRCERASE	Kombiniert die invertierten Farben von *dst* und die Farben der Quelle mit einer booleschen UND-Operation.

SRCPAINT	Kombiniert die Farben der Quelle und von *dst* mit einer booleschen ODER-Operation.
WHITENESS	Füllt *dst* mit Index-1-Farbe der physikalischen Palette (standardmäßig ist dies Weiß).

Wenn *rop* eine Operation anzeigt, die die Quell-Bitmap ausschließt, muß das Argument *srcDC* 0 sein. Ein Wert von 1 in der Maske zeigt an, daß die Ziel- und Quell-Pixelfarben unter Verwendung der höherwertigen Wortes von *rop* kombiniert werden sollen. Ein Wert von 0 in der Maske zeigt an, daß die Ziel- und Quell-Pixelfarben unter Verwendung des niederwertigen Wortes von *rop* kombiniert werden sollen. Wenn das Maskenrechteck kleiner als das von *dst* ist, wird das Maskenmuster entsprechend dupliziert.

Bei Aufzeichnung einer erweiterten Metadatei tritt ein Fehler auf, wenn der Quell-DC den erweiterten Metadatei-DC identifiziert.

Wenn eine Drehung oder Scherung für die Quell-DC beim Aufruf von *MaskBlt* vorliegt, tritt ein Fehler auf. Andere Transformationen sind erlaubt. Falls notwendig, paßt *MaskBlt* die Farbformate des Ziels und der Maske an die der Quell-Bitmap an. Vor der Verwendung von *MaskBlt* sollte eine Anwendung *GetDeviceCaps* aufrufen, um zu bestimmen, ob die Quell- und Ziel-DCs *MaskBlt* unterstützen.

MaskBlt gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::BitBlt*, *TDC::PlgBlt*, *TDC::GetDeviceCaps*, Klasse *TBitmap*, Klasse *Tpoint, Klasse TRect*

ModifyWorldTransform `bool ModifyWorldTransform(XFORM far& xform, uint32 mode);`

Ändert die aktuelle Transformation für diesen DC mit Hilfe der Argumente *xform* und *mode*. *mode* bestimmt, wie die gegebene Struktur XFORM angewandt wird. Die Werte von *mode* sind in der folgenden Tabelle aufgelistet:

Wert	Bedeutung
MWT_IDENTITY	Setzt die aktuelle Transformation mit Hilfe der Identitätsmatrix zurück. Wenn dieser Modus angegeben ist, wird die Struktur XFORM, auf die *lpXform* zeigt, ignoriert.
MWT_LEFTMULTIPLY	Multipliziert die aktuelle Transformation mit den Daten in der Struktur XFORM. (Die Daten in der Struktur XFORM sind der linke Multiplikant und die Daten der aktuellen Transformation der rechte Multiplikant.)
MWT_RIGHTMULTIPLY	Multipliziert die aktuelle Transformation mit den Daten in der Struktur XFORM. (Die Daten in der Struktur XFORM sind der rechte Multiplikant und die Daten der aktuellen Transformation der linke Multiplikant). *ModifyWorldTransform* gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::SetWorldTransform*, Struktur *XFORM*

MoveTo

```
bool MoveTo(int x, int y);
bool MoveTo(const TPoint& point);
bool MoveTo(const TPoint& point, TPoint& oldPoint);
```

Bewegt die aktuelle Position dieses DC zu den gegebenen x- und y-Koordinaten oder zum gegebenen Punkt *point*. Die dritte Version entspricht zu *::MoveToEx* und setzt die vorherige aktuelle Position in *oldPoint*. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: Klasse *TPoint*

OffsetClipRgn

```
int OffsetClipRgn(const TPoint& delta);
```

Verschiebt den Clipping-Bereich dieses DC um die in *delta* angegebenen x- und y-Offsets. Der Rückgabewert gibt den Typ des resultierenden Clipping-Bereichs wie folgt an:

Bereich	Bedeutung
COMPLEXREGION	Clipping-Bereich hat überlappende Rahmen.
ERROR	Ungültiger DC.
NULLREGION	Clipping-Bereich ist leer.
SIMPLEREGION	Clipping-Bereich hat keine überlappende Rahmen.

Siehe auch: *TDC::GetClipBox*, Klasse *TPoint*

OffsetViewportOrg

```
virtual bool OffsetViewportOrg (const TPoint& delta,
                                TPoint* oldOrg = 0);
```

Verändert den Viewport-Ursprung dieses DC relativ zu den aktuellen Werten. Die x- und y-Komponenten von *delta* werden zum vorherigen Ursprung hinzuaddiert und der resultierende Punkt wird zum neuen Viewport-Ursprung. Der vorherige Ursprung wird in *oldOrg* gesichert. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::SetViewportOrg*, *TDC::GetViewportOrg*, Klasse *TPoint*

OffsetWindowOrg

```
bool OffsetWindowOrg(const TPoint& delta, TPoint* oldOrg = 0);
```

Verändert den Fensterursprung dieses DC relativ zu den aktuellen Werten. Die x- und y-Komponenten von *delta* werden dem vorherigen Ursprung hinzugefügt und der resultierende Punkt wird zum neuen Fensterursprung. Der vorherige Ursprung wird in *oldOrg* gesichert. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::GetWindowOrg*, *TDC::SetWindowOrg*, Klasse *TPoint*

PaintRgn

```
bool PaintRgn(const TRegion& region);
```

Zeichnet (füllt) den gegebenen Bereich *region* auf diesem DC mit Hilfe des aktuell ausgewählten Pinsels. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::FillRgn*, *TDC::SelectObject*, Klasse *TRegion*

PatBlt

```
bool PatBlt (int x, int y, int w, int h, uint32 rop=PATCOPY);
bool PatBlt (const TRect& dst, uint32 rop=PATCOPY);
```

Zeichnet das gegebene Rechteck mit Hilfe des aktuell ausgewählten Pinsels für diesen DC. Das Rechteck kann über seine oberen linken Koordinaten (*x*, *y*), die Breite *w* und Höhe *h* oder über ein einzelnes Argument *TRect* angegeben werden. Der Code für die Rasteroperation *rop* bestimmt, wie Pinsel- und Oberflächenfarbe(n) kombiniert werden. Dies wird in der folgenden Tabelle erklärt:

Wert	Bedeutung
PATCOPY	Kopiert Muster in die Ziel-Bitmap.
PATINVERT	Kombiniert Ziel-Bitmap mit Mustern, wobei ein boolescher ODER-Operator verwendet wird.
DSTINVERT	Invertiert den Ziel-Bitmap.
BLACKNESS	Setzt die gesamte Ausgabe auf die binären Werte Null.
WHITENESS	Setzt die gesamte Ausgabe auf die binären Werte Eins.

Die für *rop* in dieser Funktion erlaubten Werte sind auf eine Teilmenge der vollen 256 dreistelligen Codes für Rasteroperationen begrenzt. Insbesondere kann ein Operations-Code mit *PatBlt* nicht benutzt werden, wenn er sich auf eine Quelle bezieht.

Nicht alle Geräte unterstützen die Funktion *PatBlt*. Daher sollten die Anwendungen zuerst *TDC::GetDeviceCaps* aufrufen, um die von diesem DC unterstützten Funktionen zu überprüfen.

PatBlt gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::GetDeviceCaps*, Klasse *TRect*

PathToRegion

```
HRGN PathToRegion();
```

Bei Erfolg gibt *PathToRegion* einen Bereich zurück, der von dem geschlossenen Pfad in diesem DC erzeugt wurde. Andernfalls gibt die Funktion Null zurück.

Pie

```
bool Pie(int x1, int y1, int x2, int y2, int x3, int y3,
                int x4, int y4);

bool Pie(const TRect& rect, const TPoint& start, const
                TPoint& end);
```

Verwendet die aktuell ausgewählten Stift- und Pinselobjekte zeichnet und füllt einen tortenförmigen Keil, indem sie einen elliptischen Bogen zeichnet, dessen Mittelpunkt und Endpunkte mit Linien verbunden sind. Der Mittelpunkt der Ellipse ist auch der Mittelpunkt des durch die Argumente (*x1*, *y1*)/(*x2*, *y2*) oder durch das Argument *rect* angegebenen Rechteckes. Die Start- und Endpunkte der Torte sind entweder durch (*x3*, *y3*)/(*x4*, *y4*) oder durch die Punkte *Start* und *End* angegeben. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück. Die aktuelle Position wird durch diesen Aufruf weder benutzt noch verändert.

Siehe auch: *TDC::Chord*, *TDC::Arc*, Klasse *TPoint*, Klasse *TRegion*, Klasse *TRect*

PlayMetaFile

```
bool PlayMetaFile(const TMetaFilePict& metaFile);
```

Spielt den Inhalt der gegebenen Metadatei *metaFile* auf diesen DC. Die Metadatei kann beliebig oft auf den DC gespielt werden. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::EnumMetaFile*, *TDC::PlayMetaFileRecord,* Klasse *TMetaFilePict*

PlayMetaFileRecord

```
void PlayMetaFileRecord(HANDLETABLE far& Handletable,
                        METARECORD far& metaRecord, int count);
```

Spielt die in *metaRecord* gegebene Metadateiaufzeichnung auf diesen DC. Dazu wird der GDI-Funktionsaufruf ausgeführt, der in dieser Aufzeichnung enthalten ist. *Handletable* legt die zu verwendende Objekt-Handle-Tabelle fest. *count* legt die Anzahl von Handles in der Tabelle fest.

Siehe auch: *TDC::PlayMetaFile*, *TDC::EnumMetaFile*, Struktur *HANDLETABLE*, Struktur *METARECORD*

PlgBlt

```
bool PlgBlt(const TPoint& dst, const TDC& srcDC, const TRect& src,
            const TBitmap& maskBm,  const TPoint& maskPos,
            uint32 rop=SRCCOPY);
```

Nur 32-Bit. Führt einen Bit-Blocktransfer von der gegebenen Quell-DC in diese DC durch. Farb-Bits werden vom Rechteck *src* auf *srcDC,* (dem Quell-DC), in das Parallelogramm *dst* auf diesem DC kopiert. Das Array *dst* gibt drei Punkte A, B und C als Ecken des Zielparallelogramms an. Der vierte Punkt D wird intern aus der Vektorgleichung D = B + C - A erzeugt. Die obere linke Ecke von *src* wird auf A abgebildet, die obere rechte auf B, die untere linke auf C und die untere rechte auf D. Eine optionale monochrome Bitmap kann über das Argument *maskBm* angegeben werden. (Wenn *maskBm* eine gültige monochrome Bitmap angibt, verwendet *PlgBlt* sie, um die Farb-Bits im Quellrechteck zu maskieren. Wenn *maskBm* keine monochrome Bitmap ist, tritt ein Fehler auf.) *maskPos* gibt die Koordinaten der oberen linken Ecke der

Masken-Bitmap an. Wenn *maskBm* gültig ist, bedeutet der Wert Eins in der Maske, daß das Quellfarbpixel nach *dst* kopiert wird. Der Wert Null in der Maske zeigt dagegen an, daß der entsprechende Farbpixel in *dst* nicht geändert wird. Wenn das Maskenrechteck kleiner als *dst* ist, wird das Maskenmuster entsprechend dupliziert.

Die Zielkoordinaten werden passend zu diesem DC (dem Ziel-DC) transformiert. Die Quellkoordinaten werden entsprechend dem Quell-DC transformiert. Wenn eine Drehung oder Scherung für den Quell-DC aktiv ist, wenn *PlgBlt* aufgerufen wird, tritt ein Fehler auf. Andere Transformationen wie zum Beispiel Skalierung, Verschiebung oder Spiegelung sind dagegen erlaubt. Der Streckmodus dieses DC (dem Ziel-DC) bestimmt, wie *PlgBlt* die Pixel bei Bedarf strecken oder stauchen wird. Wenn eine erweiterte Metadatei aufgezeichnet wird, tritt ein Fehler auf, sobald der Quell-DC den erweiterten Metadatei-DC identifiziert.

Falls erforderlich, paßt *PlgBlt* die Farbformate der Quelle an denen des Ziels an. Wenn die DC von Quelle und Ziel inkompatibel sind, tritt ein Fehler auf. Vor der Verwendung von *PlgBlt* sollte eine Anwendung *GetDeviceCaps* aufrufen, um zu bestimmen, ob Quell- und Ziel-DC kompatibel sind.

PlgBlt gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: Elementfunktion *TDC::GetDeviceCaps*, Elementfunktion *TDC::MaskBlt*, Elementfunktion *TDC::SetStretchBltMode*, Elementfunktion *TDC::StretchBlt*, Klasse *TBitmap*, Klasse *TPoint*, Klasse *TRect*

PolyBezier

```
bool PolyBezier(const TPoint* points, int count);
```

Zeichnet mit Hilfe des aktuell ausgewählten Stiftobjektes eine oder mehrere verbundene würfelförmige Bezier-Kurvenlinien durch die im Array *points* angegebenen Punkte. Die erste Kurvenlinie wird vom ersten zum vierten Punkt des Array gezeichnet. Dabei dienen der zweite und dritte Punkt als Kontrollelemente. Eventuell nachfolgende Kurvenlinien erfordern jeweils drei weitere Punkte im Array, da der vorherige Endpunkt als Startpunkt der nächsten Kurvenlinie genommen wird. Das Argument *count* (>= 4) gibt die gesamte Anzahl von Punkten an, die für die Angabe der vollständigen Zeichnung benötigt werden. Um n Kurvenlinien zu zeichnen, muß *count* auf $(3n + 1)$ gesetzt werden. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück. Die aktuelle Position wird bei diesem Aufruf weder benutzt noch verändert. Die resultierende Figur wird nicht gefüllt.

Siehe auch: Elementfunktion *TDC::PolyBezierTo*, Klasse *TPoint*

PolyBezierTo

```
bool PolyBezierTo(const TPoint* points, int count);
```

Zeichnet mit Hilfe des aktuell ausgewählten Stiftobjektes eine oder mehrere verbundene würfelförmige Bezier-Kurvenlinien durch die im Array

points angegebenen Punkte. Die erste Kurvenlinie wird von der aktuellen Position zum dritten Punkt des Array gezeichnet. Dabei dienen der erste und zweite Punkt als Kontrollelemente. Eventuell nachfolgende Kurvenlinien erfordern jeweils drei weitere Punkte im Array, da der vorherige Endpunkt als Startpunkt der nächsten Kurvenlinie genommen wird. Das Argument *count* (>= 4) gibt die gesamte Anzahl von Punkten an, die für die Angabe der vollständigen Zeichnung benötigt werden. Um *n* Kurvenlinien zu zeichnen, muß *count* auf (3*n*) gesetzt sein. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück. Die aktuelle Position wird bei diesem Aufruf zum Endpunkt der letzten Bezier-Kurve bewegt. Die resultierende Figur wird nicht gefüllt.

Siehe auch: *TDC::PolyBezier*, Klasse *TPoint*

PolyDraw

```
bool PolyDraw(const TPoint* points, uint8* types, int count);
```

Zeichnet mit Hilfe des aktuell ausgewählten Stiftobjektes eine oder mehrere, eventuell unverbundene Mengen von Liniensegmenten und/oder Bezier-Kurvenlinien auf diesem DC. Die *count* Punkte im Array *points* liefern die Endpunkte für jedes Liniensegment und/oder die End- und Kontrollpunkte für jede Bezier-Kurvenlinie . Die *count* BYTEs im Array *types* legen fest, wie der korrespondierende Punkt in *points* zu interpretieren ist:

Byte	Bedeutung
PT_BEZIERTO	Dieser Punkt ist ein Kontroll- oder Endpunkt für eine Bezier-Kurvenlinie. PT_BEZIERTO-Typen müssen in Dreiergruppen auftreten: die aktuelle Position ist der Bezier-Startpunkt, die ersten beiden PT_BEZIERTO-Punkte sind die Bezier-Kontrollpunkte und der dritte PT_BEZIERTO-Punkt ist der Bezier-Endpunkt, welcher zum neuen aktuellen Punkt wird. Wenn PT_BEZIERTO-Typen nicht in Dreiergruppen auftreten, tritt ein Fehler auf. Ein Endpunkt PT_BEZIERTO kann mit PT_CLOSEFIGURE Bitweise »ver-ODER-t« werden, um anzuzeigen, daß die aktuelle Figur durch das Zeichnen einer Kurvenlinie von diesem Endpunkt zum Startpunkt der letzten unverbundenen Figur geschlossen wird.
PT_CLOSEFIGURE	Optionales Flag, das wie zuvor erklärt mit PT_LINETO oder PT_BEZIERTO Bit-weise »ver-ODER-t« werden kann. Zum Abschluß wird der aktuelle Punkt auf den neuen Endpunkt umgesetzt.
PT_LINETO	Eine Linie wird von der aktuellen Position bis zu diesem Punkt, der dann zur neuen aktuellen Position wird, gezogen. PT_LINETO kann mit PT_CLOSEFIGURE Bit-weise »ver-ODER-t« werden, um anzuzeigen, daß die aktuelle Figur durch das Zeichnen eines Liniensegmentes von diesem Punkt zum Startpunkt der letzten unverbundenen Figur geschlossen wird.
PT_MOVETO	Dieser Punkt startet einen neue (unverbundene) Figur und wird zum aktuellen Punkt.

PolyDraw ist eine Alternative zu aufeinanderfolgenden Aufrufen von *MoveTo*, *LineTo*, *Polyline*, *PolyBezier* und *PolyBezierTo*. Wenn es einen durch *BeginPath* geöffneten, aktiven Pfad gibt, wird *PolyDraw* zu diesem Pfad hinzufügt.

Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::MoveTo*, *TDC::LineTo*,*TDC::PolyBezier*, *TDC::PolyBezierTo*, *TDC::Polyline*, *TDC::BeginPath*, Klasse *TPoint*

Polygon

```
bool Polygon(const TPoint* points, int count);
```

Zeichnet und füllt ein geschlossenes Polygon mit *count* (>= 2) Liniensegmenten auf diesem DC. Dabei wird der aktuell ausgewählte Stift und der Polygon-Füllmodus verwendet. Das Array *points* legt die Ecken des zu zeichnenden Polygons fest. Falls erforderlich, wird das Polygon automatisch geschlossen, indem eine Linie von der letzten zur ersten Ecke gezeichnet wird. Die aktuelle Position wird durch *Polygon* weder benutzt noch verändert. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::Polyline*, *TDC::SetPolyFillMode*, *TDC::GetPolyFillMode*, Klasse *TPoint*

Polyline

```
bool Polyline(const TPoint* points, int count);
```

Zeichnet eine Sequenz von *count* (>= 2) Liniensegmenten auf diesem DC mit Hilfe des aktuellen Stiftobjektes. Das Array *points* legt die Sequenz der zu verbindenden Punkte fest. Die aktuelle Position wird durch *Polyline* weder benutzt noch verändert. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::Polygon*, *TDC::PolyPolyline*, *TPoint*

PolylineTo

```
bool PolylineTo(const TPoint* points, int count);
```

Zeichnet ein oder mehrere verbundene Liniensegmente auf diesem DC mit Hilfe des aktuell ausgewählten Stiftobjektes. Die erste Linie wird von der aktuellen Position bis zum ersten der *count* Punkte im Array *points* gezeichnet. Eventuell nachfolgende Linien verbinden die übrigen Punkte im Array, wobei jeder Endpunkt den Startpunkt des nächsten Segmentes darstellt. Der letzte Punkt wird zum aktuellen Punkt. Es findet kein Füllen statt, selbst, wenn ein geschlossen Figur gezeichnet wird. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::PolyDraw*, *TDC::LineTo*, Klasse *TPoint*

PolyPolygon

```
bool PolyPolygon(const TPoint* points, const int* PolyCounts,
                 int count);
```

Zeichnet und füllt eine Serie von *count* (>= 2), möglicherweise überlappender, geschlossenere Polygone auf diesem DC mit Hilfe des aktuellen Stiftes und des Polygon-Füllmodus. Das Array *points* gibt die Ecken des zu zeichnenden Polygons an. *PolyCounts* ist ein Array mit *count* Ganzzahlen, die die Anzahl der Ecken in jedem Polygon angeben. Jedes Polygon muß geschlossen sein. Die aktuelle Position wird durch *Polygon* weder benutzt noch verändert. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::PolyPolyline*, *TDC::SetPolyFillMode*, *TDC::GetPolyFillMode, Klasse TPoint*

PolyPolyline

```
bool PolyPolyline(const TPoint* points,
                  const int* PolyCounts, int count);
```

Zeichnet eine Serie von *count* Polylinien (verbundene Liniensegmente) auf diesem DC mit Hilfe des aktuell ausgewählten Stiftobjektes. Die resultierenden Figuren werden nicht gefüllt. Das Array *PolyCounts* enthalten *count* Ganzzahlen, die die Anzahl der Punkte (>= 2) in jeder Polylinie angeben. Das Array *points* enthält dann alle zu verbindenden Punkte. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück. Die aktuelle Position wird durch den Aufruf weder benutzt noch verändert.

Siehe auch: *TDC::Polyline*, *TDC::PolyPolygon*, Klasse *TPoint*

PtVisible

```
bool PtVisible(const TPoint& point) const;
```

Gibt TRUE zurück, wenn der gegebe Punkt *point* innerhalb des Clipping-Bereiches dieses DC liegt. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::RectVisible*, Klasse *TPoint*

RealizePalette

```
int RealizePalette();
```

Bildet die in diesem DC ausgewählten Einträge der logischen Palette auf die Systempalette ab.

Siehe auch: Klasse *TPalette*

Rectangle

```
bool Rectangle(int x1, int y1, int x2, int y2);
bool Rectangle(const TPoint& p1, const TPoint& p2);
bool Rectangle(const TPoint& point, const TSize& s);
bool Rectangle(const TRect& rect);
```

Zeichnet und füllt ein Rechteck der gegebenen Größe auf diesem DC mit den aktuellen Stift- und Pinselobjekten. Die aktuelle Position wird durch den Aufruf weder benutzt noch verändert. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::RoundRect*, Klasse *TPoint*, Klasse *TRect*, Klasse *TSize*

RectVisible

```
bool RectVisible(const TRect& rect) const;
```

Gibt TRUE zurück, wenn irgendein Teil des gegebenen Rechteckes *rectangle* innerhalb des Clipping-Bereiches dieses DC liegt. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::PtVisible*, Klasse *TRect*

ResetDC

```
virtual bool ResetDC(DEVMODE far& devMode);
```

Aktualisiert diesen DC unter Benutzung der in der gegebenen Struktur *devMode* vorliegenden Daten. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: Struktur *DEVMODE*

RestoreBrush

```
void RestoreBrush();
```

Stellt das originale GDI-Pinselobjekt in diesem DC wieder her.

Siehe auch: *TDC::OrgBrush*, Klasse *TBrush*

RestoreDC

```
virtual bool RestoreDC(int savedDC = -1);
```

Stellt den gegebenen *savedDC* wieder her. Gibt TRUE zurück, wenn der Kontext erfolgreich wiederhergestellt worden ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::SaveDC*

RestoreFont

```
virtual void RestoreFont();
```

Stellt das originale GDI-Schriftobjekt in diesem DC wieder her.

Siehe auch: *TDC::OrgFont*, Klasse *TFont*

RestoreObjects

```
void RestoreObjects();
```

Stellt alle originalen GDI-Objekte in diesem DC wieder her.

Siehe auch: Klasse *TGdiObject*

RestorePalette

```
void RestorePalette();
```

Stellt das originale GDI-Palettenobjekt in diesem DC wieder her.

Siehe auch: *TDC::OrgPalette*, Klasse *TPalette*

RestorePen

```
void RestorePen();
```

Stellt das originale GDI-Stiftobjekt in diesem DC wieder her.

Siehe auch: *TDC::OrgPen*, Klasse *TPen*

RestoreTextBrush

```
void RestoreTextBrush();
```

Stellt das originale GDI-Textpinselobjekt in diesem DC wieder her.

Siehe auch: Klasse *TBrush*

RoundRect

```
bool RoundRect(int x1, int y1, int x2, int y2, int x3, int y3);
bool RoundRect(const TPoint& p1, const TPoint& p2,
                     const TPoint& rad);
bool RoundRect(const TPoint& p, const TSize& s, const TPoint& rad);
bool RoundRect(const TRect& rect, const TPoint& rad);
```

Zeichnet und füllt ein abgerundetes Rechteck der gegebenen Größe auf diesem DC mit den aktuellen Stift- und Pinselobjekten. Die aktuelle Position wird durch den Aufruf weder benutzt noch verändert. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::Rectangle*, Klasse *TPoint*, Klasse *TRect*, Klasse *TSize*

SaveDC

```
virtual int SaveDC() const;
```

Speichert den aktuellen Zustand dieses DC auf einem Kontext-Stack. Der gesicherte Zustand kann später mit *RestoreDC()* wiederhergestellt werden. Gibt einen Wert zurück, der den gespeicherten DC spezifiziert, oder Null, wenn der Aufruf fehlschlägt.

Siehe auch: *TDC::RestoreDC*

ScaleViewportExt

```
virtual bool ScaleViewportExt(int xNum, int xDenom, int yNum,
                                  int yDenom, TSize* oldExtent = 0);
```

Modifiziert die Viewport-Ausdehnungen des DC relativ zu den aktuellen Werten. Die neuen Ausdehnungen werden wie folgt abgeleitet:

xNewVE = (xOldVE * *xNum*)/ *xDenom*

yNewVE = (yOldVE * *yNum*)/ *yDenom*

Die vorherigen Ausdehnungen werden in *oldExtent* gespeichert. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::SetViewportExt*, Klasse *TSize*

ScaleWindowExt

```
virtual bool ScaleWindowExt(int xNum, int xDenom, int yNum,
                                int yDenom, TSize* oldExtent = 0);
```

Modifiziert die Fenster-Ausdehnungen des DC relativ zu den aktuellen Werten. Die neuen Ausdehnungen werden wie folgt abgeleitet:

xNewWE = (xOldWE * *xNum*)/ *xDenom*
yNewWE = (yOldWE * *yNum*)/ *yDenom*

Die vorherigen Ausdehnungen werden in *oldExtent* gespeichert. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::SetWindowExt*, *TSize*

ScrollDC

```
bool ScrollDC(int x, int y, const TRect& scroll, const TRect&
                    clip, TRegion& updateRgn, TRect& updateRect);
```

ScrollDC

```
bool ScrollDC(const TPoint& delta, const TRect& scroll,
                    const TRect& clip, TRegion& updateRgn,
                    TRect& updateRect);
```

Blättert ein Rechteck mit Bits horizontal um *x* (oder *delta.x* in der zweiten Version) Geräteeinheiten und vertikal um *y* (oder *delta.y*) Geräteeinheiten auf diesem DC. Die Rechtecke zum Blättern und Clipping sind durch *scroll* und *clip* angegeben. *ScrollDC* stellt im Argument *updateRgn* Daten bereit, die den Bereich (nicht notwendig rechteckig) beschreiben, die durch das Blättern unbetroffen blieb. Auf ähnliche Weise beschreibt *ScrollDC* in *updateRect* das Rechteck (in Client-Koordinaten), das den Aktualisierungsbereich zum Blättern begrenzt. Dies ist das größte Feld, das ein Neuzeichnen erfordert.

Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: Klasse *TPoint*, Klasse *TRect*, Klasse *TRegion*

SelectClipPath

```
bool SelectClipPath(int mode);
```

Wählt den aktuellen Pfad auf diesem DC als einen Clipping-Bereich aus. Dabei wird der Pfad mit existierenden Clipping-Bereichen mit Hilfe des gegebenen Modus *mode*, wie in folgender Tabelle gezeigt, kombiniert:

Modus	Bedeutung
RGN_AND	Der neue Clipping-Bereich umfaßt die überlappenden Felder des aktuellen Clipping-Bereichs und des aktuellen Pfades (Schnittmenge).
RGN_COPY	Der neue Clipping-Bereich ist der aktuelle Pfad.
RGN_DIFF	Der neue Clipping-Bereich umfaßt die Felder des aktuellen Clipping-Bereichs ohne die des aktuellen Pfades.
RGN_OR	Der neue Clipping-Bereich umfaßt die kombinierten Felder des aktuellen Clipping-Bereichs und des aktuellen Pfades (Vereinigung).
RGN_XOR	Der neue Clipping-Bereich umfaßt die kombinierten Felder des aktuellen Clipping-Bereichs und des aktuellen Pfades, allerdings ohne die überlappenden Felder.

Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

SelectClipRgn

```
int SelectClipRgn(const TRegion& region);
```

Wählt den gegebenen Bereich *region* als den aktuellen Clipping-Bereich für diesen DC aus. Eine Kopie des gegebenen Bereichs *region* wird benutzt, damit Sie denselben Bereich für andere DC-Objekte auswählen kön-

nen. Der Rückgabewert zeigt den Typ des neuen Clipping-Bereichs wie folgt an:

Bereich	Bedeutung
COMPLEXREGION	Clipping-Bereich hat überlappende Rahmen.
ERROR	Ungültiger DC.
NULLREGION	Clipping-Bereich ist leer.
SIMPLEREGION	Clipping-Bereich hat keine überlappende Rahmen.

Siehe auch: *TDC::OffsetClipRgn*, *TDC::GetClipBox*, Klasse *TRegion*

SelectObject

```
void SelectObject(const TBrush& brush);
```

```
void SelectObject(const TPen& pen);
virtual void SelectObject(const TFont& font);
void SelectObject(const TPalette& palette, bool forceBackground = false);
```

Wählt das gegebene GDI-Objekt in diesem DC aus. Das vorher ausgewählte Objekt wird in dem geeigneten Datenelement *OrgXXX* gespeichert. Wenn für ein Palettenargument *forceBackgound* auf FALSE gesetzt ist (der Vorgabewert), stellt die ausgewählte logische Palette eine Vordergrundpalette dar, sobald das Fenster den Eingabefokus besitzt. Wenn *forceBackground* TRUE ist, ist die ausgewählte logische Palette immer eine Hintergrundpalette, dies unabhängig davon, ob das Fenster den Fokus besitzt oder nicht.

Siehe auch: *TDC::OrgBrush*, *TDC::OrgFont*, *TDC::OrgPalette*, *TDC::OrgPen*, *TDC::OrgTextBrush*, Klasse *TBrush*, Klasse *TFont*, Klasse *TPalette*, Klasse *TPen*, *TMemoryDC::SelectObject*

SelectStockObject

```
virtual void SelectStockObject(int index);
```

Wählt für einen DC einen vordefinierten Stift, Pinsel, eine vordefinierte Schrift oder Palette aus.

Siehe auch: *TPrintPreviewDC::SelectStockObject*

SetBkColor

```
virtual TColor SetBkColor(TColor color);
```

Setzt die aktuelle Hintergrundfarbe dieses DC auf den gegebenen Wert *color* oder den nächsten möglichen Farbwert. Gibt 0x80000000 zurück, wenn der Aufruf fehlschlägt.

Siehe auch: *TDC::GetBkColor*, Klasse *TColor*

SetBkMode

```
int SetBkMode(int mode);
```

Setzt den Hintergrundmodus auf das gegebene Argument *mode*, d.h. OPAQUE oder TRANSPARENT. Gibt den vorherigen Hintergrundmodus zurück.

Siehe auch: *TDC::GetBkMode*

SetBoundsRect

```
uint SetBoundsRect(TRect& bounds, uint flags);
```

Kontrolliert die Ansammlung von Grenzrechteckinformation für diesen DC. Abhängig vom Wert von *flags* kann sich das Rechteck *bounds* (möglicherweise NULL) mit dem existierenden angesammelten Rechteck kombinieren oder dieses ersetzen. *flags* kann jede passende Kombination der folgenden Werte sein:

Konstante	Bedeutung
DCB_ACCUMULATE	Füge *bounds* (rechteckige Vereinigung) zum aktuell angesammelten Rechteck hinzu.
DCB_DISABLE	Schalte die Ansammlung von Grenzen ab.
DCB_ENABLE	Schalte die Ansammlung von Grenzen an (Die Voreinstellung für die Ansammlung von Grenzen ist abgeschaltet.).
DCB_RESET	Setze das Grenzrechteck auf leer.
DCB_SET	Setze das Grenzrechteck auf *bounds*.

Es gibt zwei Ansammlung von Grenzrechtecken, eine für Windows und eine für die Anwendung. Die für Windows angesammelten Grenzen können durch eine Anwendung zwar abgefragt, aber nicht verändert werden. Die Anwendung kann dagegen die angesammelten Grenzen des DC sowohl abfragen als auch ändern.

Siehe auch: *TDC::GetBoundsRect*, Klasse *TPoint*

SetBrushOrg

```
bool SetBrushOrg(const TPoint& origin, TPoint* oldOrg = 0);
```

Setzt den Ursprung des aktuell ausgewählten Pinsels dieses DC mit dem gegebenen Wert *origin*. Der vorherige Ursprung wird an *oldOrg* übergeben. Gibt bei erfolgreichem Aufruf TRUE zurück. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::GetBrushOrg*, Klasse *TPoint*

SetDIBits

```
bool SetDIBits(TBitmap& bitmap, uint16 startScan, uint16 numScans,
                    const void HUGE* bits, const BITMAPINFO far& Info,
                    uint16 usage);
bool SetDIBits(TBitmap& Bitmap, const TDib& dib);
```

Die erste Version setzt die Pixel in *bitmap* (die gegebene Ziel-Bitmap auf diesem DC), mit den Farbdaten der Quell-DIB (DIB = Device-Independent Bitmap, Geräte-unabhängige Bitmap), die sich im Byte-Array *bits* und der BITMAPINFO-Struktur *Info* befinden. *numScan* Zeilen werden von der DIB beginnend bei der Zeile *startScan* übernommen. Das Argument *usage* gibt an, wie das Element *bmiColors* von BITMAPINFO interpretiert wird. Dies wird bei *TDC::GetDIBits()* erläutert.

In der zweiten Version von *SetDIBits* werden die Pixel in *bitmap* mit dem gegebenen Quell-Argument *TDib* gesetzt.

SetDIBits gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::GetDIBits*, *TDC::SetDIBitsToDevice*, Klasse *TBitmap*, Struktur *BITMAPINFO*, Klasse *TDib*

SetDIBitsToDevice

```
bool SetDIBitsToDevice(const TRect& dst, const TPoint& src,
                       uint16 startScan, uintq16 numScans,
                       const void HUGE* bits,
                       const BITMAPINFO far& bitsInfo, uint16
usage);
bool SetDIBitsToDevice(const TRect& dst, const TPoint& src,
                       const TDib& dib);
```

Die erste Version setzt die Pixel in *dst* (dem gegebenen Zielrechteck auf diesem DC) mit den Farbdaten der Quell-DIB (DIB = Device-Independent Bitmap, Geräte-unabhängige Bitmap), die sich im Byte-Array *bits* und der BITMAPINFO-Struktur *bitsInfo* befinden. Der DIB-Ursprung wird durch den Punkt *src* angegeben. *numScan* Zeilen werden von der DIB beginnend bei Zeile *startScan* übernommen. Das Argument *usage* gibt an, wie das Element *bmiColors* von BITMAPINFO interpretiert wird. Dies wird bei *TDC::GetDIBits()* erläutert.

In der zweiten Version von *SetDIBitsToDevice* werden die Pixel in *dst* mit dem gegebenen Quell-Argument *TDib* gesetzt.

SetDIBitsToDevice gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::GetDIBits*, Klasse *TDib*, Klasse *TPoint*, Klasse *TRect*, Struktur *BITMAPINFO*

SetMapMode

```
int SetMapMode(int mode);
```

Setzt den aktuellen Fenster-Abbildungsmodus für diesen DC auf *mode*. Gibt den Wert des vorherigen Abbildungsmodus zurück. Der Abbildungsmodus definiert, wie logische Koordinaten in Gerätekoordinaten abgebildet werden. Er kontrolliert ferner die Orientierung der x- und y-Achsen des Gerätes. Siehe *TDC::GetMapMode* für eine vollständige Liste der Abbildungsmodi.

Siehe auch: *TDC::GetMapMode*

SetMapperFlags

```
uint32 SetMapperFlags(uint32 flag);
```

Verändert den von der Schriftabbildung verwendeten Algorithmus, wenn logische Schriften in physikalische Schriften auf diesem DC abgebildet werden. Bei Erfolg setzt die Funktion das aktuelle Flag für Schriftabbildung auf *flag* und gibt das vorherige Abbildungs-Flag zurück. Andernfalls wird GDI_ERROR zurückgegeben. Das Abbildungs-Flag bestimmt, ob die Schriftabbildung das XY-Verhältnis einer Schrift an das von diesem

DC anpassen wird. Wenn das Bit 0 von *flag* auf Eins gesetzt ist, wählt die Abbildung nur übereinstimmende Schriften aus. Wenn keine übereinstimmenden Schriften existieren, wird ein neues XY-Verhältnis gewählt und eine Schrift ermittelt, die zu diesem Verhältnis paßt.

SetMiterLimit

```
bool SetMiterLimit(float newLimit, float* oldLimit = 0);
```

Setzt die Grenze der Winkelverbindungen auf *newLimit* und legt den vorherigen Wert in *oldLimit* ab. Gibt TRUE zurück, wenn er Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

SetPixel

```
TColor SetPixel(int x, int y, TColor color);
TColor SetPixel(const TPoint& p, TColor color);
```

Setzt die Farbe des Pixel an der gegebenen Position auf die gegebene Farbe und gibt die vorherige Pixel-Farbe zurück.

Siehe auch: *TDC::GetPixel, TColor, TPoint*

SetPolyFillMode

```
int SetPolyFillMode(int mode);
```

Setzt den Polygon-Füllmodus für diesen DC auf den gegebenen Moduswert, entweder ALTERNATE oder WINDING. Gibt den vorherigen Füllmodus zurück.

Siehe auch: *TDC::GetPolyFillMode, TDC::Polygon*

SetROP2

```
int SetROP2(int mode);
```

Setzt den aktuellen Vordergrund-Mischmodus dieses DC auf den gegebenen Wert *mode* und gibt den vorherigen Modus zurück. Das Argument *mode* bestimmt, wie Pinsel, Stift und das existierende Bildschirmbild beim Füllen und Zeichnen kombiniert werden. *mode* kann einen der folgenden Werte annehmen:

Wert	Bedeutung
R2_BLACK	Pixel ist immer binär Null.
R2_COPYPEN	Pixel ist die Stiftfarbe.
R2_MASKNOTPEN	Pixel ist eine Kombination der Farben, die der Anzeige und dem Inversen des Stiftes gemeinsam sind.
R2_MASKPEN	Pixel ist eine Kombination der Farben, die der Anzeige und dem Stift gemeinsam sind.
R2_MASKPENNOT	Pixel ist eine Kombination der Farben, die dem Inversen der Anzeige und dem Stift gemeinsam sind.
R2_MERGEPEN	Pixel ist eine Kombination der Stiftfarbe und der Anzeigefarbe.
R2_MERGENOTPEN	Pixel ist eine Kombination der Anzeigefarbe und des Inversen der Stiftfarbe.
R2_MERGEPENNOT	Pixel ist eine Kombination der Stiftfarbe und des Inversen der Anzeigefarbe.

R2_NOP	Pixel bleibt unverändert.
R2_NOT	Pixel ist das Inverse der Anzeigefarbe.
R2_NOTCOPYPEN	Pixel ist das Inverse der Stiftfarbe.
R2_NOTMASKPEN	Pixel ist das Inverse der R2_MASKPEN-Farbe.
R2_NOTMERGEPEN	Pixel ist das Inverse der R2_MERGEPEN-Farbe.
R2_NOTXORPEN	Pixel ist das Inverse der R2_XORPEN-Farbe.
R2_WHITE	Pixel ist immer binär Eins.
R2_XORPEN	Pixel ist eine Kombination der Farben im Stift und in der Anzeige, aber nicht in beiden.

Siehe auch: *TDC::GetROP2*, *TDC::GetDeviceCaps*

SetStretchBltMode

```
int SetStretchBltMode(int mode);
```

Setzt den Streckmodus dieses DC auf den gegebenen Wert *mode* und gibt den vorherigen Modus zurück. Das Argument *mode* definiert, welche Zeilen und/oder Spalten durch *TDC::StretchBlt()* eliminiert werden: BLACKONWHITE, COLORONCOLOR oder WHITEONBLACK.

Siehe auch: *TDC::GetStretchBltMode*, *TDC::StretchBlt*

SetSystemPaletteUse

```
int SetSystemPaletteUse(int usage);
```

Ändert die Verwendung der Systempalette dieses DC. Das Argument *usage* kann SYSPAL_NOSTATIC oder SYSPAL_STATIC sein. Gibt den vorherigen Wert für die Verwendung zurück.

Siehe auch: *TDC::GetSystemPaletteUse*

SetTextAlign

```
uint SetTextAlign(uint flags);
```

Setzt die Flags für die Textausrichtung für diesen DC. Bei Erfolg gibt *SetTextAlign* die vorherigen Flags für die Textausrichtung zurück. Andernfalls gibt die Funktion GDI_ERROR zurück. Die Flag-Werte gleichen denen für die Funktion *TDC::GetTextAlign* aufgelisteten. Die Flags für die Textausrichtung bestimmen, wie *TDC::TextOut()* und *TDC::ExtTextOut* Text-Strings im Verhältnis zur Bildschirmposition des ersten Zeichens ausrichten.

Siehe auch: *TDC::GetTextAlign*, *TDC::TextOut*, *TDC::ExtTextOut*

SetTextCharacterExtra

```
int SetTextCharacterExtra(int extra);
```

Bei Erfolg setzt die Funktion den Zeichenabstand für diesen DC auf *extra* (in logischen Einheiten) und gibt den vorherigen Zeichenabstand zurück. Andernfalls gibt die Funktion Null zurück. Wenn der aktuelle Abbildungsmodus *nicht* MM_TEXT ist, wird der Wert *extra* auf das nächste Pixel hin transformiert und gerundet.

Siehe auch: *TDC::GetTextCharacterExtra*

SetTextColor

```
virtual TColor SetTextColor(TColor color);
```

Setzt die aktuelle Textfarbe dieses DC auf den gegebenen Wert *color*. Die Textfarbe bestimmt die von *TDC::TextOut()* und *TDC::ExtTextOut()* angezeigte Farbe.

Siehe auch: *TDC::GetTextColor*, Klasse *TColor*

SetTextJustification

```
bool SetTextJustification(int breakExtra, int breakCount);
```

Setzt *breakExtra* logische Einheiten als absoluten Extrazwischenraum, der auf *breakCount* Break-Zeichen hinzuaddiert werden soll, wenn Text-Strings mit *TDC::TextOut()* und *TDC::ExtTextOut()* angezeigt werden. Der Extrazwischenraum wird auch zwischen Break-Zeichen zugeteilt. Das Break-Zeichen ist üblicherweise das Zeichen mit ASCII-Code 32 (Leerzeichen). Einige Schriften definieren allerdings dafür auch andere Zeichen. *TDC::GetTextMetrics()* kann zum Ermitteln des Wertes des Break-Zeichens verwendet werden. *TDC::GetTextExtentPoint()* muß aufgerufen werden, um die Textbreite vor der Anpassung zu erhalten. Von daher kann der Wert *breakExtra* vor dem Aufruf von *SetTextJustification* berechnet werden.

Wenn der aktuelle Abbildungsmodus nicht MM_TEXT ist, wird der Wert extra auf das nächste Pixel hin transformiert und gerundet.

SetTextJustification gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

SetViewportExt

```
virtual bool SetViewportExt(const TSize& extent,
                            TSize* oldExtent = 0);
```

Setzt die x- und y-Ausdehnungen des Viewport dieses DC auf die gegebenen Werte *extent*. Die vorherigen Ausdehnungen werden in *oldExtent* gespeichert. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück. Der Wert *extent* bestimmt den benötigten Umfang an Strecken oder Stauchen im logischen Koordinatensystem für eine Anpassung an das Koordinatensystem des Gerätes. *extent* bestimmt ferner die relative Orientierung der beiden Koordinatensysteme.

Siehe auch: *TDC::GetViewportExt*, Klasse *TSize*

SetViewportOrg

```
virtual bool SetViewportOrg(const TPoint& origin,
                            TPoint* oldOrg = 0);
```

Setzt den Viewport-Ursprung dieses DC auf den gegebenen Wert *origin* und speichert den vorherigen Ursprung in *oldOrg*. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::GetViewportOrg*, *TDC::OffsetViewportOrg*, Klasse *TPoint*

SetWindowExt

```
virtual bool SetWindowExt(const TSize& extent, TSize* oldExtent
                                    = 0);
```

Setzt die x- und y-Ausdehnungen des Fensters dieses DC auf die gegebenen Werte *extent*. Die vorherigen Ausdehnungen werden in *oldExtent* gespeichert. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Anderenfalls gibt die Funktion FALSE zurück. Der Wert *extent* bestimmt den benötigten Umfang an Strecken oder Stauchen im logischen Koordinatensystem für eine Anpassung an das Koordinatensystem des Gerätes. *extent* bestimmt ferner die relative Orientierung der beiden Koordinatensysteme.

Siehe auch: *TDC::GetWindowExt*, *TDC::ScaleWindowExt*, Klasse *TSize*

SetWindowOrg

```
bool SetWindowOrg(const TPoint& origin, TPoint* oldOrg = 0);
```

Setzt den Ursprung des mit diesem DC assoziierten Fensters auf den gegebenen Wert *origin* und speichert den vorherigen Ursprung in *oldOrg*. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::GetWindowOrg*, *TDC::OffsetWindowOrg*, Klasse *TPoint*

SetWorldTransform

```
bool SetWorldTransform(XFORM far& xform);
```

Nur 32-Bit. Setzt für diesen DC eine zweidimensionale Lineartransformation zwischen Weltkoordinaten und Seitenkoordinaten, die durch die Struktur *xform* gegeben ist. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::ModifyWorldTransform*, Struktur *XFORM*

StretchBlt

```
bool StretchBlt(int dstX, int dstY, int dstW, int dstH,
                      const TDC& srcDC, int srcX, int srcY,
                      int srcW, int srcH, uint32 rop=SRCCOPY);
bool StretchBlt(const TRect& dst, const TDC& srcDC,
                      const TRect& src, uint32 rop=SRCCOPY);
```

Kopiert eine Bitmap von der Quell-DC in das Zielrechteck auf diesem DC, welches durch die Koordinaten der oberen linken Ecke (*dstX*, *dstY*), die Breite *dstW* und Höhe *dstH* oder (in der zweiten Version) durch ein *TRect*-Objekt *dst* gegeben ist. Die Quell-Bitmap ist in ähnlicher Form mit (*srcX*, *srcY*), *srcW* und *srcH* oder durch ein *TRect*-Objekt *src* gegeben. *StretchBlt* streckt oder staucht die Quelle entsprechend dem aktuell in diesem DC (dem Ziel-DC) eingestellten Streckmodus. Der Code für die Rasteroperation *rop* gibt an, wie die Farben bei Ausgabeoperationen, die mit einem Pinsel, einer Quell-Bitmap und einer Ziel-Bitmap arbeiten, kombiniert werden. Die Codes von *rop* sind bei der Beschreibung von *TDC::MaskBlt* erläutert.

StretchDIBits

```
bool StretchDIBits(const TRect& dst, const TRect& src,
                   const void HUGE* bits,
                   const BITMAPINFO far& bitsInfo, uint16 usage,
                   uint32 rop=SRCCOPY);
bool StretchDIBits(const TRect& dst, const TRect& src,
                   const TDib& dib, uint32 rop=SRCCOPY);
```

Kopiert die Farbdaten von *src*, dem Quellrechteck mit Pixel in der gegebenen DIB (DIB = Device-Independent Bitmap, Geräte-unabhängige Bitmap) auf diesem DC, nach *dst*, dem Zielrechteck. Die Bits und Farbdaten der DIB sind entweder im Byte-Array *bits* zusammen mit der BITMAPINFO-Struktur *bitsInfo* oder in dem *TDib*-Objekt *dib* angegeben. Die Zeilen und Spalten der Farbdaten werden gestreckt und gestaucht, um der Größe des Zielrechteckes angepaßt zu werden. Das Argument *usage* legt fest, wie das Element *bmiColors* von BITMAPINFO interpretiert wird. Dies ist bei *TDC::GetDIBits()* erläutert. Der Code für die Rasteroperation *rop* legt fest, wie die Quell-Pixel, der aktuelle Pinsel für diesen DC und die Ziel-Pixel, kombiniert werden, um ein neues Bild zu erzeugen. Siehe bei *TDC::MaskBlt* für eine detaillierte Liste von *rop*-Codes.

Siehe auch: *TDC::GetDIBits, TDC::MaskBlt*, Klasse *TDib*, Klasse *TRect*, Struktur *BITMAPINFO*

StrokeAndFillPath

```
bool StrokeAndFillPath();
```

Nur 32-Bit. Schließt alle offenen Figuren im aktuellen Pfad für diesen DC, zeichnet den Umriß des Pfades mit dem aktuellen Stift und füllt dessen Innenfläche mit Hilfe des aktuellen Pinsels und des Polygon-Füllmodus. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::StrokePath, TDC::BeginPath, TDC::FillPath, TDC::EndPath, TDC::SetPolyFillMode*, Klasse *TBrush*, Klasse *TPen*

StrokePath

```
bool StrokePath();
```

Nur 32-Bit. Gibt den aktuellen, geschlossenen Pfad auf diesem DC mit Hilfe des für den DC aktuellen Stiftes wieder.

Siehe auch: *TDC::StrokeAndFillPath, TDC::BeginPath*

TabbedTextOut

```
bool TabbedTextOut(const TPoint& p, const char far* string,
                   int count, int numPositions,
                   const int far* positions, int tabOrigin);

virtual bool TabbedTextOut(const TPoint& p, const char far* string,
                   int count, int numPositions,
                   const int far* positions, int tabOrigin,
                   TSize& size);
```

Zeichnet bis zu *count* Zeichen des gegebenen nullterminierten Strings *string* in der aktuellen Schrift auf diesem DC. Wenn *count* -1 ist, wird der gesamte String geschrieben.

Tabulatoren werden entsprechend den gegebenen Argumenten expandiert. Das Array *positions* gibt *numPositions* Spaltenformate in Geräteeinheiten an. Die Spaltenformate müssen im Array streng anwachsende Werte haben. Wenn sowohl *numPositions* als auch *positions* Null sind, werden Tabulatoren auf das achtfache einer durchschnittlichen Zeichenbreite expandiert. Wenn *numPositions* Eins ist, werden die Spaltenformate *positions[0]* Einheiten voneinander getrennt. *tabOrigin* gibt die x-Koordinate in logischen Einheiten an, von der ab die Expandierung von Tabulatoren startet.

Das Argument *p* legt die logischen Koordinaten des Referenzpunktes fest, der für die Ausrichtung des ersten Zeichens abhängig vom aktuellen Text-Ausrichtungsmodus benutzt wird. Dieser Modus kann mit *TDC::GetTextAlign* nachgesehen und mit *TDC::SetTextAlign* geändert werden. Standardmäßig wird die aktuelle Position durch *TabbedTextOut* weder benutzt noch verändert. Wenn der Ausrichtungsmodus allerdings auf TA_UPDATECP gesetzt ist, ignoriert *TabbedTextOut* die/das Referenzpunkt-Argument(e) und verwendet bzw. aktualisiert die aktuelle Position als den Referenzpunkt.

Das Argument *size* in der zweiten Version von *TabbedTextOut* gibt die Dimensionen (*size.y* = Höhe und *size.y* = Breite) des Strings in logischen Einheiten wieder.

TabbedTextOut gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::TextOut*, *TDC::GetTextAlign*, TDC::SetTextAlign, Klasse *TPoint*, Klasse *TSize*

TextOut

```
virtual bool TextOut(int x, int y, const char far* string,
                     int count = -1);
bool TextOut(const TPoint& p, const char far* string, int count
             = -1);
```

Zeichnet bis zu *count* Zeichen des gegebenen nullterminierten String *string* in der aktuellen Schrift auf diesen DC. Wenn *count* -1 ist (der Vorgabewert), wird der gesamte String geschrieben.

Die Argumente (*x*, *y*) oder *p* legen die logischen Koordinaten des Referenzpunktes fest, der für die Ausrichtung des ersten Zeichens abhängig vom aktuellen Text-Ausrichtungsmodus benutzt wird. Dieser Modus kann mit *TDC::GetTextAlign* nachgesehen und mit *TDC::SetTextAlign* geändert werden. Standardmäßig wird die aktuelle Position durch *TextOut* weder benutzt noch verändert. Der Ausrichtungsmodus kann allerdings

auf TA_UPDATECP gesetzt werden, wodurch Windows die aktuelle Position benutzt und aktualisiert. In diesem Modus ignoriert *TextOut* die/das Referenzpunkt-Argument(e).

TextOut gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::ExtTextOut*, *TDC::GetTextAlign*, *TDC::SetTextAlign*, Klasse *TPoint*

TextRect

```
bool TextRect(int x1, int y1, int x2, int y2);
```

```
bool TextRect(const TRect& rect);
bool TextRect(int x1, int y1, int x2, int y2, TColor color);
bool TextRect(const TRect rect, TColor color);
```

Füllt das gegebene Rechteck durch den Aufruf von *::ExtTextOut* mit einem ETO_OPAQUE-Rechtecktyp als Argument und einem NULL-String. Wenn kein Farbargument bereitgestellt wird, wird die aktuelle Hintergrundfarbe verwendet. Wenn ein Farbargument bereitgestellt wird, wird diese Farbe als aktuelle Hintergrundfarbe gesetzt, die dann für das Füllen benutzt wird. *TextRect* gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::SetBkColor*, Klasse *TColor*, Klasse *TRect*

UpdateColors

```
void UpdateColors();
```

Aktualisiert den Client-Bereich dieses DC, indem die aktuellen Farben in dem Client-Bereich an die Systempalette auf Pixel-zu-Pixel-Basis angepaßt werden.

WidenPath

```
bool WidenPath();
```

Nur 32-Bit. Redefiniert den aktuellen, geschlossenen Pfad auf diesem DC als den Bereich, der gemalt wird, wenn der Pfad mit dem aktuellen Stift dieses DC gezeichnet wird. Der aktuelle Stift muß unter den folgenden Bedingungen erzeugt worden sein:

Wenn die Konstruktoren *TPen int Style, int Width, TColor Color)* oder *TPen(const LOGPEN* LogPen)* benutzt wurden, muß die Breite des Stiftes in Geräteeinheiten größer als Eins sein.

Wenn die Konstruktoren *TPen(uint32 PenStyle, uint32 Width, const TBrush& Brush, uint32 StyleCount, LPDWORD pSTyle)* oder *TPen(uint32 PenStyle, uint32 Width, const LOGBRUSH& logBrush, uint32 StyleCount, LPDWORD pSTyle)* benutzt wurden, muß der Stift ein geometrischer Stift sein.

Alle Bezier-Kurven in dem Pfad werden in Sequenzen linearer Segmente konvertiert, die eine Näherung zu erweiterten Kurven darstellen. Damit

bleiben nach dem Aufruf von *WidenPath* keine Bezier-Kurven mehr im Pfad.

WidenPath gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDC::FlattenPath*, *TDC::BeginPath*, Klasse *TPen*

Geschützte Konstruktoren

Konstruktor

```
TDC();
```

Nur für die Verwendung durch abgeleitete Klassen. Ruft *Init* auf, um die Datenelemente *OrgXXX* zu löschen und setzt *ShouldDelete* auf TRUE.

Konstruktor

```
TDC(HDC handle, TAutoDelete AutoDelete);
```

Nur für die Verwendung durch abgeleitete Klassen. Erzeugt unter Verwendung eines bestehenden DC-Handles ein TDC-Objekt. Ruft *Init* auf, um die Datenelemente *OrgXXX* zu löschen.

Siehe auch *TDC::Init*

Geschützte Datenelemente

Handle

```
TGdiBase::Handle;
```

Das Handle dieses DC. Verwendet das Handle derBasisklasse (*TGdiBase::Handle*).

Siehe auch: *TDC*

OrgBrush

```
HBRUSH OrgBrush;
```

Handle für das originale GDI-Pinselobjekt für diesen DC. Enthält das vorherige Pinselobjekt immer dann, wenn mit *SelectObject(brush)* ein neuer Pinsel ausgewählt wurde.

Siehe auch: *TDC::SelectObject*, Klasse *TBrush*

OrgFont

```
HFONT OrgFont;
```

Handle für das originale GDI-Schriftobjekt für diesen DC. Enthält das vorherige Schriftobjekt immer dann, wenn mit *SelectObject(font)* eine neue Schrift ausgewählt wurde.

Siehe auch: *TDC::SelectObject*, Klasse *TFont*

OrgPalette

```
HPALETTE OrgPalette;
```

Handle für das originale GDI-Palettenobjekt für diesen DC. Enthält das vorherige Palettenobjekt immer dann, wenn mit *SelectObject(palette)* eine neue Palette ausgewählt wurde.

Siehe auch: *TDC::SelectObject*, Klasse *TPalette*

OrgPen

```
HPEN OrgPen;
```

Handle für das originale GDI-Stiftobjekt für diesen DC. Enthält das vorherige Stiftobjekt immer dann, wenn mit *SelectObject(pen)* ein neuer Stift ausgewählt wurde.

Siehe auch: *TDC::SelectObject*, Klasse *Tpen*

OrgTextBrush

```
HBRUSH OrgTextBrush;
```

Nur 32-Bit. Handle für das originale GDI-Textpinselobjekt für diesen DC. Speichert das vorherige Textpinsel-Handle immer dann, wenn ein neuer Pinsel ausgewählt wurde mit *SelectObject(text_brush)*.

Siehe auch: *TDC::SelectObject*, Klasse *TBrush*

ShouldDelete

```
TGdiBase::ShouldDelete;
```

Auf TRUE gesetzt, wenn das Handle für dieses Objekt durch den Destruktor gelöscht werden sollte. Andernfalls auf FALSE gesetzt.

Geschützte Elementfunktionen

CheckValid

```
TGdiBase::CheckValid (uint resId=IDS_GDIFAILURE)
```

CheckValid

```
static void CheckValid(HANDLE handle, uint resId=IDS_GDIFAILURE)
```

Beide Versionen von *CheckValid* überprüfen, ob es ein gültiges GDI-Objekt-Handle gibt. Wird ein solches nicht gefunden, wird eine GDI-Exception für die angegebene Ressourcen-Kennung ausgelöst. Beide Versionen verwenden *TGdiBase::CheckValid*.

GetAttributeHDC

```
virtual HDC GetAttributeHDC() const;
```

Gibt die Attribute des DC-Objektes zurück.

GetHDC

```
HDC GetHDC() const;
```

Gibt ein Handle auf den DC zurück.

Init

```
void Init();
```

Setzt *OrgBrush*, *OrgPen*, *OrgFont*, *OrgBitmap* und *OrgPalette* auf Null und setzt *ShouldDelete* auf TRUE. Diese Funktion dient nur internen Zwecken der TDC-Konstruktoren.

Siehe auch: *TDC Konstruktoren*, *TDC::SelectObject*

Klasse TDecoratedFrame decframe.h

TDecoratedFrame positioniert sein Client-Fenster automatisch (Ein Client-Fenster muß bereitgestellt werden), so daß es dieselbe Größe wie das Client-Rechteck besitzt. Sie können zusätzliche Dekorationen wie Mauspalette und Statuszeilen einem Fenster hinzufügen. Sie können einen *TDecoratedFrame* ohne Titelleiste erzeugen, in dem Sie alle Bits im Datenelement *style* der Struktur *TWindowAttr* löschen. *TDecoratedFrame* ist eine streamfähige Klasse.

Verwenden Sie *TOleFrame* für OLE-fähige Anwendungen, um einen dekorierten Rahmen zu erzeugen und Dekorationen wie beispielsweise Mauspaletten für das Hauptfenster einer SDI (Einzel-Dokument-Schnittstelle) OLE-Anwendung zu verwalten.

Siehe auch Klasse *TOleFrame*

Typdefinitionen

TLocation

```
enum TLocation (Top, Bottom, Left, Right);
```

Der Aufzählungstyp *TLocation* beschreibt die Positionen *Top, Left, Bottom* und *Right*, an die die Dekoration gesetzt werden kann. *Insert* verwendet diesen Aufzählungstyp, um die Dekoration zu positionieren.

Öffentlicher Konstruktor

Konstruktor

```
TDecoratedFrame(TWindow* parent, const char far *title, TWindow*
                clientWnd, bool trackMenuSelection = false, TModule*
                module = 0);
```

Konstruiert ein *TDecoratedFrame*-Objekt mit dem angegebenen übergeordneten Fenster (*parent*), dem Fenstertitel (*title*) und der Modul-ID. Setzt *Attr.Title* auf den neuen Titel. Übergibt einen Zeiger auf das Client-Fenster, sofern einer angegeben wurde. Mit FALSE als Voreinstellung kontrolliert *trackMenuSelection*, ob Hinweistext am unteren Rand des Fensters erscheint, wenn ein Menüeintrag hervorgehoben wird.

Öffentliche Elementfunktionen

Insert

```
void Insert (TWindow& decoration, TLocation = Top);
```

Nachdem Sie angeben, wo die Dekoration plaziert werden soll, fügt sie *Insert* oben, unten links oder rechts vom Client-Fenster hinzu. Dieser Prozeß ist insbesondere dann wichtig, wenn es mehrere Dekorationen gibt. *Insert* inspiziert das Datenelement *Attr.Style* der Dekoration und überprüft das Flag WS_VISIBLE, um zu klären, ob die Dekoration anfangs

sichtbar oder versteckt sein soll. Die Position der Dekoration *Insert* verwendet *TLocation enum*, welches die Positionen *Top*, *Left*, *Bottom* und *Right* für die Plazierung der Dekoration beschreibt.

PreProcessMsg

```
bool PreProcessMsg (MSG& msg);
```

Überschreibt die in *TFrameWindow* definierte virtuelle Funktion, damit Dekorationen die Möglichkeit erhalten, die Vorverarbeitung für mnemonischen Zugriff durchzuführen.

Siehe auch: *TFrameWindow::PreProcessMsg*, *TWindow::PreProcessMsg*

SetClientWindow

```
TWindow* SetClientWindow(TWindow* clientWnd);
```

Überschreibt die in *TFrameWindow* definierte virtuelle Funktion. Setzt das Client-Fenster auf das angegebene Fenster. Der Anwender muß das alte Client-Fenster selbst entfernen, wenn er dies möchte.

Geschützte Datenelemente

MenuItemId

```
uint MenuItemID;
```

Gibt die Menüeintrags-ID an.

TrackMenuSelection

```
bool TrackMenuSelection;
```

Gibt an, ob Informationen zur Menüauswahl und zum Hilfestatus sichtbar sein sollen.

Geschützte Elementfunktionen

EvCommand

```
LRESULT EvCommand(uint Id, HWND hWndCtl, uint notifyCode);
```

Automatisiert das Verstecken und Zeigen von Dekorationen.

EvCommandEnable

```
void EvCommandEnable(TCommandEnabler& ce);
```

Verwaltet das Markieren und das Aufheben der Markierung bei Menüeinträgen, die mit Dekorationen assoziiert sind.

EvEnterIdle

```
void EvEnterIdle(uint source, HWND hWndDlg);
```

Antwortet auf eine Windows-API-Botschaft WM_ENTERIDLE, die dem Hauptfenster einer Anwendung mitteilt, daß ein Dialogfenster oder ein Menü in den Idle-Zustand eintritt. *EvEnterIdle* verwaltet ferner die Aktualisierung der Statuszeile mit der passenden Hilfemeldung.

EvMenuSelect

```
void EvMenuSelect(uint MenuItemId, UINT flags, HMENU hMenu);
```

Antwortet auf eine Benutzermenüauswahl. Wenn *MenuItemId* leer ist, zeigt die Funktion eine leere Hilfemeldung an. Andernfalls zeigt sie eine Hilfemeldung mit der angegebenen String-ID an. Siehe *EvEnterIdle* für eine Beschreibung darüber, wie die Hilfemeldung geladen wird.

EvSize

```
inline void EvSize(uint sizeType, TSize& size);
```

Übergibt eine WM_SIZE-Botschaft an *TLayoutWindow*.

SetupWindow

```
void SetupWindow();
```

Ruft *TLayoutWindow::Layout* auf, um die Größe und Position der Dekoration festzulegen.

Siehe auch: *TFrameWindow::SetUpWindow*, *TWindow::SetUpWindow*, *TLayoutWindow::Layout*

Anworttabelleneinträge

Anworttabelleneintrag	Elementfunktion
EV_WM_ENTERIDLE	EvEnterIdle
EV_WM_MENUSELECT	EvMenuSelect
EV_WM_SIZE	EvSize

Klasse TDecoratedMDIFrame — decmdifr.h

TDecoratedMDIFrame ist ein MDI-Rahmen (MDI = Multiple Document Interface, Multidokumentschnittstelle), der dekorierte untergeordnete Fenster unterstützt und sowohl von *TMDIFrame* wie auch von *TDecoratedFrame* abgeleitet ist.

TDecorated MDIFrame unterstützt benutzerdefinierte Mauspaletten. Sie können einen Satz Dekorationen (beispielsweise Mauspaletten und Lineale) in einen dekorierten Rahmen einfügen. Wird ein anderer Satz Werkzeuge benötigt, kann der bisherige Satz entfernt und ein anderer Satz von Dekorationen neu eingefügt werden. Stellen Sie jedoch sicher, daß alle unerwünschten Dekorationen von den angepaßten Seiten (d. h., oben, links, unten und rechts) entfernt wurden, bevor Sie einen neuen Satz einfügen.

TDecoratedMDIFrame ist eine streamfähige Klasse.

Öffentliche Konstruktoren

Konstruktor

```
TDecoratedMDIFrame(const char far *title, TResId menuResId,
                   TMDIClient &clientWnd = *new TMDIClient,
                   bool trackMenuSelection = false, TModule* module = 0);
```

Konstruiert einen dekorierten MDI-Rahmen (MDI = Multiple Document Interface, Multidokumentschnittstelle) des angegebenen Client-Fensters

mit der angezeigten Menü-Ressourcen-ID. Standardmäßig wird Menü-Hinweistext nicht angezeigt.

Geschützte Elementfunktion

DefWindowProc

```
LRESULT DefWindowProc(uint message, WPARAM wParam, LPARAM lParam);
```

Überschreibt *TWindow:DefWindowProc*. Wenn der Botschafts-Parameter WM_SIZE ist, gibt *DefWindowProc* 0 zurück. Andernfalls gibt *DefWindowProc* das Ergebnis des Aufrufs von *TMDIFrame::DefWindowProc* zurück.

Siehe auch: *TMDIFrame::DefWindowProc*

Anworttabelleneinträge

Die Antworttabelle von *TDecoratedMDIFrame* hat keine Einträge.

Klasse TDesktopDC — dc.h

Die von *TWindowDC* abgeleitete DC-Klasse (DC = Device Context, Gerätekontext) *TDesktopDC* bietet Zugriff auf den Client-Bereich des Desktop-Fensters, der den Bildschirm hinter allen anderen Fenstern darstellt.

Öffentlicher Konstruktor

Konstruktor

```
TDesktopDC();
```

Standardkonstruktor für *TDesktopDC*-Objekte.

Klasse TDialog — dialog.h

Ein Dialogfenster, das normalerweise seine Informationen vom Anwender erhält, ist ein Fenster, das andere Dialogelemente wie beispielsweise Schalter oder Bildlaufleisten aufnehmen kann. Anders als wirklich untergeordnete Fenster, die nur im Client-Bereich des übergeordneten Fensters dargestellt werden können, sind Dialogfenster auf dem ganzen Bildschirm frei beweglich. *TDialog*-Objekte repräsentieren sowohl modale als auch nicht-modale Dialogfenster als Oberflächenelemente. (Ein modales Dialogfenster unterbindet Operationen in seinem übergeordneten Fenster, solange es geöffnet ist. Sie arbeiten daher in nur einem Windows-"Modus".)

Ein *TDialog*-Objekt besitzt eine korrespondierende Ressourcen-Definition, die die Plazierung und Erscheinungsform seiner Dialogelemente be-

schreibt. Der Bezeichner dieser Ressourcen-Definition wird an den Konstruktor des *TDialog*-Objektes übergeben. Ein *TDialog*-Objekt ist mit einem modalen oder nicht-modalen Oberflächenelement assoziiert, indem dessen Elementfunktion *Execute* bzw. *Create* aufgerufen wird.

Sie können *TDialog* zum Erstellen einer Anwendung, die ein Dialogfenster als ihr Haupfenster besitzt, verwenden. Dazu muß das Dialogfenster als *TDialog* erzeugt werden und als Client eines *TFrameWindow* übergeben werden. Der Programmcode hierzu könnte wie folgt aussehen:

```
SetMainWindow(new TFrameWindow(0, "title" new TDialog(0, IDD_MYDIALOG)));
```

ObjectWindows bietet eine dreidimensionale (3-D) Unterstützung für Dialogfenster. Wenn Ihre Anwendung eine Verwendung der DLL CTL3D von Microsoft vorsieht, müssen Sie Ihre Anwendung durch einen Aufruf von *TApplication::EnableCtl3d* registrieren. ObjectWindows wird dann die Botschaft WM_CTLCOLOR automatisch an die DLL CTL3D weiterleiten.

ObjectWindows bietet ferner BWCC-Unterstützung (BWCC = Borland Custom Controls library, Bibliothek für Dialogelemente im Borland-Stil) für Dialogfenster. Solange kein selbstdefiniertes Template angegeben wird, verwendet *TDialog* die BWCC-Templates. (Standardmäßig aktiviert die Elementfunktion *EnableBWCC* von *TApplication* diese BWCC-Unterstützung.)

TDialog ist eine streamfähige Klasse.

ObjectWindows bietet außerdem Verkapselung für allgemeine Dialogfenster, die es dem Anwender ermöglichen, die Schriftart, den Dateinamen, Farbe, Druckoptionen usw. auszuwählen. *TCommonDialog* ist die übergeordnete Klasse für diese Gruppe von allgemeinen Dialogfenster-Klassen.

Öffentliche Datenelemente

Attr

```
TDialogAttr Attr;
```

Attr enthält die Attribute des Dialogfensters zur Dialogerzeugung (beispielsweise Größe und Layout).

IsModal

```
bool IsModal;
```

IsModal ist TRUE, wenn das Dialogfenster modal ist, und FALSE, wenn es nicht-modal ist.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TDialog(TWindow* parent, TResId resId, TModule* module = 0);
```

Ruft einen *TWindow*-Kostruktor auf, übergibt dabei *parent* und *module* und ruft *DisableAutoCreate* auf, damit das *TDialog*-Objekt nicht automa-

tisch zusammen mit seinem übergeordneten Fenster erzeugt und angezeigt wird. *TDialog* initialisiert dann *Title* mit -1 und setzt *Attr.Name* mit Hilfe des Ganzzahl- oder String-Wertes für den Ressourcen-Bezeichner dieses Dialogs. Dieser Bezeichner muß mit einer Dialog-Ressourcen-Definition in der Ressourcendatei korrespondieren. Abschließend initialisiert die Funktion *Attr.Param* mit Null und setzt *IsModal* auf FALSE.

Destruktor

```
~TDialog();
```

Wenn *Attr.Name* den Ressourcen-Bezeichner in Form eines Strings (keine Ganzzahl) angibt, gibt dieser Destruktor den reservierten Speicher, der den Namen von *TDialog* (*Attr.Name*) enthielt, wieder frei.

Siehe auch TApplication*::EnableBWCC*, *TWindow::~TWindow*, *TWindow::DisableAutoCreate*, *TWindow::TWindow*, *TDialog::Attr*

Öffentliche Elementfunktionen

CloseWindow

```
void CloseWindow(int retValue = IDCANCEL);
```

Schließt mit Bedingungen das Dialogfenster. Wenn das Dialogfenster nicht-modal ist, ruft die Funktion *TWindow::CloseWindow* auf. Wenn das Dialogfenster modal ist, ruft sie *CanClose* auf. Wenn *CanClose* TRUE zurückgibt, ruft *CloseWindow TransferData* auf, um die Daten des Dialogfensters zu transferieren, und übergibt der Funktion dabei *retValue*. Der Vorgabewert von *retValue* ist IDCANCEL.

Siehe auch: *TWindow::CloseWindow*

CmCancel

```
void CmCancel();
```

Automatische Antwort auf einen Klick auf den »Abbruch«-Schalter des Dialogfensters. Ruft *CloseWindow* auf und übergibt IDCANCEL.

Siehe auch: *TDialog::CloseWindow*

CmOk

```
void CmOk();
```

Antwortet auf einen Klick auf den »OK«-Schalter des Dialogfensters (mit dem Bezeichner IDOK). Ruft *CloseWindow* auf und übergibt IDOK.

Siehe auch: *TDialog::CloseWindow*

Create

```
virtual bool Create();
```

Erzeugt ein nicht-modales Dialogfenster als Oberflächenelement, das mit dem *TDialog*-Objekt assoziiert ist. Ruft *DisableAutoCreate* auf, um die automatische Erzeugung aller untergeordneten Fenster zu verhindern. Ruft *EnableKBHandler* auf, um die Tastaturbehandlung zu aktivieren. Registriert alle untergeordneten Fenster des Dialogs für die Unterstützung selbstdefinierter Dialogelemente. Ruft die Windows-Funktion *CreateDialogParam* auf, um das Dialogfenster zu erzeugen.

Create gibt TRUE zurück, wenn er Aufruf erfolgreich ist. Wenn er nicht erfolgreich ist, löst *Create* eine *TXInvalidWindow*-Exception aus.

Siehe auch: *TDialog::Execute*, *TModule::MakeWindow*, *TWindow::DisableAutoCreate*

Destroy

```
virtual void Destroy(int retValue = IDCANCEL);
```

Entfernt das Oberflächenelement, das mit dem *TDialog*-Objekt assoziiert ist. Wenn das Element ein nicht-modales Dialogfenster ist, ruft *Destroy TWindow::Destroy* auf. Wenn das Element ein modales Dialogfenster ist, ruft *Destroy EnableAutoCreate* bei allen untergeordneten Fenstern auf. Danach ruft *Destroy* die Windows-Funktion *::EndDialog* auf und übergibt dabei *retValue*, der als Rückgabewert dient, um das Ergebnis der Dialogausführung anzuzeigen. Der Vorgabewert für *retValue* ist IDCANCEL.

Siehe auch: *TWindow::Destroy*, *TWindow::EnableAutoCreate*

DialogFunction

```
virtual bool DialogFunction(uint message, WPARAM wParam, LPARAM lParam);
```

Um Botschaften innerhalb der Dialogfunktion zu verarbeiten, muß Ihre Anwendung diese Funktion überschreiben. *DialogFunction* gibt TRUE zurück, wenn die Botschaft behandelt worden ist, und FALSE, wenn die Botschaft nicht behandelt worden ist.

DoCreate

```
virtual HWND DoCreate();
```

Create ruft *DoCreate* auf, um die aktuelle Erzeugung eines Dialogfensters durchzuführen.

DoExecute

```
virtual int DoExecute();
```

Execute ruft *DoExecute* auf, um für die aktuelle Ausführung eines Dialogfensters zu sorgen.

Siehe auch ~~TDialog::Execute~~

EvClose

```
void EvClose();
```

Antwortet auf einen ankommende *EvClose*-Botschaft mit dem Schließen des Fensters.

EvCtlColor

```
HBRUSH EvCtlColor(HDC hDC, HWND hWndChild, uint ctlType);
```

Übergibt das Handle auf den Bildschirmkontext für das untergeordnete Fenster, das Handle auf das übergeordnete Fenster und die Standardsystemfarben für das übergeordnete Fenster. Das übergeordnete Fenster benutzt dann das Bildschirmkontext-Handle (gegeben in *hDC*), um die Farben des untergeordneten Fensters für Text und Hintergrund zu setzen.

Wenn dreidimensionale (3-D) Unterstützung aktiviert ist, behandelt *EvCtlColor* die EV_WM_CTLCOLOR-Botschaft, indem sie der DLL CTL3D die Verarbeitung der WM_CTLCOLOR-Botschaft erlaubt, um die Hintergrundfarbe zu setzen und einen Hintergrundpinsel für das Fenster bereitzustellen.

Siehe auch *TApplication::EnableCtl3d*

EvInitDialog

```
virtual bool EvInitDialog(HWND hWndFocus);
```

EvInitDialog wird automatisch kurz vor der Anzeige des Dialogfensters aufgerufen. Die Funktion ruft *SetupWindow* auf, um alle für das Dialogfenster oder dessen Dialogelemente erforderlichen Einstellungen durchzuführen.

Siehe auch: *TWindow::SetupWindow*

EvPaint

```
void EvPaint();
```

EvPaint ruft die allgemeine *TWindow* Standard-Verarbeitungsfunktion *DefaultProcessing* für das von Windows unterstützte Zeichnen auf.

Siehe auch: *TWindow::DefaultProcessing*

EvSetFont

```
virtual void EvSetFont(HFONT hfont, bool redraw);
```

Antwortet auf eine Anforderung, die Schrift eines Dialoges zu ändern.

Execute

```
virtual int Execute();
```

Execute erzeugt ein modales Dialogfenster als Oberflächenelement, das mit dem *TDialog*-Objekt assoziiert ist, und führt es aus. Wenn das Element erfolgreich assoziiert werden konnte, kehrt *Execute* solange nicht zurück, bis *TDialog* geschlossen wurde.

Execute führt die folgenden Aufgaben aus:

Registriert die Fensterklasse des Dialogfensters und alle untergeordneten Fenster des Dialogfensters.

Ruft *DoExecute* auf, um das Dialogfenster auszuführen.

Prüft, ob Exceptions aufgetreten sind und löst eine *TXWindow*-Exception aus, wenn ein Fehler auftritt.

Execute gibt einen Integerwert zurück, der angibt, wie der Anwender das modale Dialogfenster geschlossen hat. Kann das Dialogfenster nicht erzeugt werden, gibt *Execute* -1 zurück.

Siehe auch: *TModule::ExecDialog*, *TWindow::DisableAutoCreate*, *TXWindow*

GetDefaultId

```
uint GetDefaultId() const;
```

Liefert die Standard-Ressourcen-ID.

GetItemHandle

```
HWND GetItemHandle(int childId);
```

Gibt das Fenster-Handle des Dialogfenster-Dialogelementes zurück, das durch die mitgelieferte ID identifiziert wird. Da *GetItemHandle* nun veraltet ist, sollten neue Anwendungen *TWindow::GetDlgItem* verwenden.

PerformDlgInit

```
bool PerformDlgInit();
```

Initialisiert die Dialogelemente mit dem Inhalt von RT_DLGINIT. Dies ist der Ressoucen-Bezeichner, der die Erscheinung und die Lage der Steuerelemente (Schalter, Gruppenfelder, usw.) beschreibt. Gibt bei Erfolg TRUE zurück. Andernfalls wird FALSE zurückgegeben.

PreProcessMsg

```
bool PreProcessMsg(MSG & msg);
```

Überschreibt die virtuelle Funktion, die von *TWindow* definiert ist, um die Vorverarbeitung von Fensterbotschaften durchzuführen. Wenn das untergeordnete Fenster die Tastatursteuerung angefordert hat, behandelt PreProcessMsg alle Tastenkürzelbotschaften und verarbeitet dann alle anderen Tastaturbotschaften.

Siehe auch *TWindow::PreProcessMsg*, Struktur *MSG*

SendDlgItemMsg

```
uint32 SendDlgItemMsg(int ChildId, uint16 Msg, uint16 WParam, uint32
LParam);
```

Sendet eine Windows-Dialogelementbotschaft, die durch *Msg* identifiziert wird, an das Dialogelement des Dialogfensters, das durch seine mitgelieferte ID *ChildID* identifiziert wird. *WParam* und *LParam* werden in der Dialogelementbotschaft zu Parametern. *SendDlgItemMsg* gibt den vom Dialogelement zurückgegebenen Wert zurück. Wenn die Dialogelement-ID ungültig ist, gibt die Funktion Null zurück. Diese Funktion ist veraltet.

SetCaption

```
void SetCaption(const char far* title);
```

Setzt den Titel des Dialogfensters auf den Wert des Parameters *title*.

Siehe auch: *TWindow::SetCaption*

SetDefaultId

```
void SetDefaultId(uint Id);
```

Setzt die Standard-Ressourcen-ID.

Geschützte Elementfunktionen

GetClassName

```
char far* GetClassName();
```

GetClassName überschreibt die in *TWindow* definierte virtuelle Funktion und gibt den Namen der Windows-Standardklasse für das Dialogfenster zurück. Diese Klasse muß für ein modales Dialogfenster verwendet werden. Für ein nicht-modales Dialogfenster gibt *GetClassName* den Namen des *TWindow*-Standardobjektes zurück. Wenn BWCC (BWCC = Borland Custom Controls library, Bibliothek für Dialogelemente im Borland-Stil) aktiviert ist, gibt *GetClassName* BORDLGCLASS zurück.

Siehe auch: *TWindow::GetClassName*

GetWindowClass

```
void GetWindowClass(WNDCLASS& wndClass);
```

Überschreibt die in *TWindow* definierte virtuelle Funktion. Füllt *WndClass* mit den Registrierungsattributen eines *TDialog*-Objektes, die von einem

existierenden *TDialog*-Fenster oder von BWCC (falls aktiviert) erhalten werden.

Wenn das Layout der Klasse mit CS_GLOBALCLASS registriert ist, muß die Registrierung des Klassen-Layouts aufgehoben werden. Dies geschieht durch Ausschalten des Layout-Bits. Beispielsweise:

```
{
baseclass::GetWindowClass(wndClass);
...
WndClass.style&=~CS_GLOBALCLASS:
...
}
```

Siehe auch: *TWindow::GetWindowClass*, Klasse *TWindow*, Struktur *WNDCLASS*

SetupWindow

```
void SetupWindow();
```

Überschreibt die in *TWindow* definierte virtuelle Funktion. Richtet das Dialogfenster ein, indem *SetCaption* (setzt den Titel *Title*) und *TWindow::SetupWindow* aufgerufen werden.

Wenn die dreidimensionale (3-D) Unterstützung aktiviert ist, ruft *SetupWindow* die DLL CTL3D auf, um das Dialogfenster zu erzeugen.

Siehe auch: *TCommonDialog::SetupWindow*, *TDialog::SetCaption*, *TWindow::SetupWindow*

Anworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_COMMAND (IDCANCEL, CmCancel)	CmCancel
EV_COMMAND (IDOK, CmOk)	CmOk
EV_CTLCOLOR	EvCtlColor
EV_WM_CLOSE	EvClose
EV_WM_PAINT	EvPaint

Siehe auch Klasse *TCommonDialog*

Klasse TDialog dialog.h

Ein *TDialogAttr*-Objekt wird benutzt, um die Attribute zur Erzeugung eines *TDialog*-Objektes wie beispielsweise das Layout, das Erscheinungsbild, die Größe und den Typ der Dialogelemente, die zum Dialogfenster

gehören, zu speichern. *TDialogAttr* enthält zwei Datenelemente: *Name* (die Ressourcen-ID) und *Param*. Diese Elemente enthalten anwenderdefinierte Daten, die bei der Erzeugung des Dialogfensters verwendet werden.

Öffentliche Datenelemente

Name

```
char far* Name;
```

Name enthält den Bezeichner der Dialog-Ressource, der entweder ein String oder ein Integer-Ressourcen-Bezeichner sein kann.

Param

```
uint32 Param;
```

Param enthält den Parameter, der an das Dialogfenster bei dessen Erzeugung übergeben wird. Sie können diesem Feld einen Wert in einem Konstruktor einer abgeleiteten Klasse übergeben. Obwohl Informationen vom Typ *Param*, die dem Dialogfenster übergeben werden, als Datenelemente gespeichert werden können, ist *Param* besonders dann hilfreich, wenn Sie ein Dialogfenster erzeugen wollen, das durch Programmcode, der nicht unter ObjectWindows geschrieben wurde, implementiert wird.

Nachdem *Param* akzeptiertiert wurde, steht der Wert in den mit WM_INITDIALOG assoziierten Botschaftsantwortfunktionen (beispielsweise *EvInitDialog*) zur Verfügung.

Siehe auch: *TDialog::Attr*

Klasse TDib — gdiobjec.h

Die Klasse *TDib* ist von *TGdiObject* abgeleitet und repräsentiert GDI-DIB-Objekte (DIB = Device-Independent Bitmap, geräteunabhängige Bitmap). *TDibDC* kapselt die Erzeugung von DCs mit Hilfe von DIB.DRV (ein GDI-Treiber, der mit Windows-MME und 3.1 bereitgestellt wird). DIBs besitzen kein Windows-Handle; sie sind nur Strukturen, die Format- und Paletteninformationen sowie eine Kollektion von Bits oder Pixel enthalten. *TDib* bietet einen bequemen Weg, mit DIBs wie mit allen anderen GDI-Objekten zu arbeiten. Der Speicher für die DIB liegt in einer mit *GlobalAlloc* reservierten Einheit, wodurch er an die Zwischenablage, an OLE 2 (= Object Linking and Embeding) usw. übergeben werden kann.

Der Destruktor von *TDib* überlädt den Basisdestruktor, da DIBs keine realen GDI-Objekte sind.

Typdefinitionen

Map

```
enum Map(MapFace, MapText, MapShadow, MapHighlight, MapFrame);
```

Zählt die Werte für den Teil des Fensters auf, dessen Farbe gesetzt werden soll. Sie können mit ODER verbunden werden, um die Farben der Schrift auf Aktionsschaltern, die Farbe eines ausgewählten Dialogschalters, die Kantenschattierung auf Aktionsschaltern, Text auf Aktionsschaltern, die Farbe des Fensterrahmens und die Hintergrundfarbe der unterschiedlichen Teile eines Fensters zu steuern. Die Funktion *MapUIColors* verwendet einen dieser Werte, um die Farben der unterschiedlichen Teile eines Fensters auf einer angegebenen Farbe abzubilden.

Siehe auch *TDib::MapUIColors*

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TDib(HGLOBAL handle, TAutoDelete autoDelete = NoAutoDelete);
```

Erzeugt ein *TDib*-Objekt und setzt das Datenelement *Handle* auf das gegebene, entliehene *handle*. Das Datenelement *ShouldDelete* nimmt standardmäßig FALSE an, um sicherzustellen, daß das entliehene Handle nicht gelöscht wird, wenn das C++ Objekt entfernt wird.

Konstruktor

```
TDib(const TClipboard& clipboard);
```

Konstruiert ein *TDib*-Objekt mit einem Handle, das von der gegebenen Zwischenablage entliehen ist.

Konstruktor

```
TDib(const TDib& dib);
```

Dieser öffentliche Kopierkonstruktor erzeugt eine vollständige Kopie des gegebenen *dib*-Objektes, wie in:

```
TDib myDib = yourDib;
```

Konstruktor

```
TDib(int width, int height, int nColors, uint16 mode = DIB_RGB_COLORS);
```

Erzeugt ein DIB-Objekt mit den gegebenen Werten Breite, Höhe, Anzahl der Farben und Modus.

Konstruktor

```
TDib(HINSTANCE instance, TResID resID);
```

Erzeugt ein DIB-Objekt aus der Ressource mit der gegebenen ID.

Konstruktor

```
TDib(const char* name);
```

Erzeugt ein DIB-Objekt aus der gegebenen Ressourcendatei.

Konstruktor

```
TDib(const TBitmap& bitmap, const TPalette* pal = 0);
```

Erzeugt ein DIB-Objekt aus der gegebenen Ressource-Bitmap und Palette. Wenn *pal* den Wert Null hat (Voreinstellung), wird die Standardpalette verwendet.

Destruktor

```
~TDib();
```

Überschreibt den Basisdestruktor.

Siehe auch *::GetClipboardData, ~TGdiObject, TDib::InfoFromHandle, TDib::LoadFile, TDib::LoadResource, TGdiObject::Handle, TGdiObject::ShouldDelete*

Öffentliche Elementfunktionen

operator BITMAPINFO()

```
operator const BITMAPINFO far*() const;
operator BITMAPINFO far*();
```

Wandelt diesen DIB-Typ um, indem ein Zeiger auf die Struktur der Bitmap-Information (BITMAPINFO), die Angaben zum Farbformat und den Abmessungen (Größe, Breite, Höhe, Auflösung usw.) dieser DIB enthält, zurückgegeben wird.

Siehe auch: *TDib::GetInfo*, Struktur *BITMAPINFO*

operator BITMAPINFOHEADER()

```
operator const BITMAPINFOHEADER far*() const;
operator BITMAPINFOHEADER far*();
```

Wandelt diesen DIB-Typ um, indem ein Zeiger auf den Bitmap-Info-Header dieser DIB zurückgegeben wird.

Siehe auch: *TDib::GetInfoHeader*, Struktur *BITMAPINFOHEADER*

ChangeModeToPal

```
bool ChangeModeToPal(const TPalette& pal);
```

Konvertiert eine existierende Farbtabelle an Stelle der Verwendung von palettenbezogenen Werten. Die übergebene Palette dient als Referenz. Ist der Aufruf erfolgreich, wird **true**zurückgegeben, andernfalls **false**.

Siehe auch: *TDib::ChangeModeToRGB*, *TPalette::GetPaletteEntry*

ChangeModeToRGB

```
bool ChangeModeToRGB(const TPalette& pal);
```

Konvertiert eine existierende Farbtabelle an Stelle der Verwendung absoluter RGB-Werte. Die übergebene Palette dient als Referenz. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDib::ChangeModetoPal*, *TPalette::GetPaletteEntry*

FindColor

```
int FindColor(TColor color);
```

Gibt den zur gegebenen Farbe korrespondierenden Paletteneintrag zurück.

Siehe auch: *TDib::GetColor*, *TColor*, *TDib::SetColor*, *TDib::MapColor*

FindIndex

```
int FindIndex(uint16 index);
```

Gibt den zum gegebenen Index korrespondierenden Paletteneintrag zurück.

Siehe auch: *TDib::GetIndex*, *TDib::SetIndex*, *TDib::MapIndex*

GetBits

```
const void HUGE* GetBits() const;
void HUGE* GetBits();
```

Gibt das Datenelement *Bits* für diese DIB zurück.

Siehe auch: *TDib::Bits*

GetColor

```
TColor GetColor(int entry) const;
```

Gibt die Farbe für den gegebenen Paletteneintrag zurück.

Siehe auch: *TDib::SetColor*, *TColor*, *TDib::FindColor*, *TDib::MapColor*

GetColors

```
inline const TRgbQuad far* GetColors() const;
TRgbQuad far* GetColors();
```

Gibt den Wert *bmiColors* für diese DIB zurück.

Siehe auch: *TRgbQuad*

GetIndex

```
uint16 GetIndex(int entry) const;
```

Gibt den Farbindex für den gegebenen Paletteneintrag zurück.

Siehe auch: *TDib::SetIndex*, *TDib::FindIndex*,, *TDib::MapIndex*

GetIndices

```
inline const uint16 far* GetIndices() const;
uint16 far* GetIndices();
```

Gibt die *bmiColors*-Indizes für diese DIB zurück.

Siehe auch: *TDib::GetColors*

GetInfo

```
const BITMAPINFO far* GetInfo() const;
BITMAPINFO far* GetInfo();
```

Gibt das Feld *Info* dieser DIB zurück. Die Struktur BITMAPINFO einer DIB enthält Informationen über die Abmessungen und die Farbe der DIB und legt ein Array von Datentypen fest, die die Farben in der Bitmap definieren.

Siehe auch: *TDib::Info*, *TDib::GetInfoHeader*

GetInfoHeader

```
const BITMAPINFOHEADER far* GetInfoHeader() const;
BITMAPINFOHEADER far* GetInfoHeader();
```

Gibt *bmiHeader* dieser DIB zurück. Dieses Feld der Struktur BITMAPINFO einer DIB enthält Informationen über die Abmessungen und die Farbe dieser DIB.

Siehe auch: *TDib::Info*, *TDib::GetInfo*, Struktur BITMAPINFOHEADER

operator HANDLE()

```
operator HANDLE() const;
```

Wandelt diesen DIB-Typ um, indem ihr *Handle* zurückgegeben wird.

Siehe auch: *TGdiObject::Handle*

Height

```
int Height() const;
```

Gibt *H*, die Höhe dieser DIB, zurück.

Siehe auch: *TDib::Info*

IsOK

bool IsOK() const;

Gibt FALSE zurück, wenn *Info* Null ist. Andernfalls gibt die Funktion TRUE zurück. Wenn beim Konstruieren der DIB ein Problem auftaucht,

wird der Speicher freigegeben und *Info* auf Null gesetzt. Deshalb stellt *Info* einen zuverlässigen Weg dar zu bestimmen, ob die DIB korrekt konstruiert wurde.

Siehe auch: *TDib* Konstruktoren, *TDib::Info*

IsPM

```
bool IsPM() const;
```

Gibt TRUE zurück, wenn *IsCore* TRUE ist; d.h., wenn die DIB eine PM-DIB im alten Stil ist, die Kern-Header verwendet. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDib::IsCore*

MapColor

```
int MapColor(TColor fromColor, TColor toColor, bool doAll = false);
```

Bildet in der aktuellen Palette dieser DIB *fromColor* in *toColor* ab. Gibt den Paletteneintrag für die gegebene Farbe zurück. Gibt den Paletteneintrag für das Argument *toColor* zurück.

Siehe auch: *TDib::GetColor*, *TColor*, *TDib::SetColor*, *TDib::FindColor*

MapIndex

```
int MapIndex(uint16 fromIndex, Word toIndex, bool doAll = false);
```

Bildet in der aktuellen Palette dieser DIB *fromIndex* in *toIndex* ab. Gibt den Paletteneintrag für das Argument *toIndex* zurück.

Siehe auch: *TDib::FindIndex*, *TDib::SetIndex*, *TDib::GetIndex*

MapUIColors

```
void MapUIColors(uint mapColors, TColor* bkColor = 0);
```

Bildet die Farben der Benutzeroberfläche auf dem im Parameter *mapColors* angegebenen Wert ab. Dieser Wert ist einer der Werte desAufzählungstyps *Map*. Verwenden Sie diese Funktion für die Farben der Schrift auf Aktionsschaltern, die Farbe eines ausgewählten Dialogschalters, die Kantenschattierung auf Aktionsschaltern, Text auf Aktionsschaltern, die Farbe des Fensterrahmens und die Hintergrundfarbe.

Siehe auch Aufzählungstyp *TDib::Map*

NumColors

```
long NumColors() const;
```

Gibt *NumClrs*, die Anzahl der Farben in der Palette dieser DIB, zurück.

Siehe auch: *TDib::Info*

NumScans

```
uint16 NumScans() const;
```

Gibt die Anzahl der Zeilen in dieser DIB zurück.

Siehe auch: *TDib::StartScan*

operator <<

```
TClipboard& operator<<(TClipboard& clipboard, TDib& dib);
```

Schreibt die gegebene *dib* in die gegebene Zwischenablage *clipboard*. Gibt eine Referenz auf die resultierende Zwischenablage zurück, wodurch ein normales Verketten von **<<** ermöglicht wird.

Siehe auch: *TClipboard*

operator ==

```
bool operator ==(const TDib& other) const;
```

Vergleicht zwei Handle und gibt TRUE zurück, wenn das Handle dieser DIB gleich dem Handle der anderen DIB (*other*) ist.

Siehe auch *TDib::Handle*

SetColor

```
void SetColor(int entry, TColor color);
```

Setzt die gegebene Farbe für den gegebenen Paletteneintrag.

Siehe auch: *TDib::GetColor*, *TColor*, *TDib::MapColor*, *TDib::FindColor*

SetIndex

```
void SetIndex(int entry, uint16 index);
```

Setzt den gegebenen Index für den gegebenen Eintrag.

Siehe auch: *TDib::GetIndex*, *TDib::FindIndex*, *TDib::MapIndex*

Size

```
TSize Size() const;
```

Gibt *TSize(W,H)*, die Größe dieser DIB zurück.

Siehe auch: *TDib::Info*, *TSize*

StartScan

```
uint16 StartScan() const;
```

Gibt die Startzeile der DIB zurück.

Siehe auch: *TDib::numScans*

ToClipboard

```
void ToClipboard(TClipboard& clipboard);
```

Legt diese DIB in der angegebenen Zwischenablage ab.

Siehe auch: *TClipboard*

operator TRgbQuad()

```
operator const TRgbQuad far*() const;
operator TRgbQuad far*() const;
```

Wandelt diesen DIB-Typ um, indem ein Zeiger auf ihre Farbstruktur zurückgegeben wird.

Siehe auch: *TDib::GetColors*, *TRgbQuad*

Usage

```
uint16 Usage() const;
```

Gibt den Modus *Mode* für diese DIB zurück. Dieser Wert teilt *GDI* mit, wie die Farbtabelle zu behandeln ist.

Siehe auch: *TDib::Mode*

Width

```
int Width() const;
```

Gibt *W*, die Breite der DIB, zurück.

Siehe auch: *TDib::Info*

WriteFile

```
bool WriteFile(const char* filename);
```

Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Geschützte Datenelemente

Bits

```
void HUGE* Bits;
```

Bits zeigt in den Speicherblock, auf den *Info* zeigt.

Siehe auch: *TDib::GetBits*

H

```
int H;
```

Die Höhe dieser DIB in Pixel.

Siehe auch: *TDib::Height*, *TDib::Size*, *TDib::NumScans*

Info

```
BITMAPINFO far* Info;
```

Gesperrter, global reservierter Block.

Siehe auch: *TDib::GetInfo*

IsCore

```
bool IsCore;
```

Ist auf TRUE gesetzt, wenn diese DIB eine PM-DIB im alten Stil ist, die Kern-Header benutzt. Andernfalls ist das Datenelement auf FALSE gesetzt.

Siehe auch: *TDib::isPM*

IsResHandle

```
bool IsResHandle;
```

Ist auf TRUE gesetzt, wenn diese DIB ein Ressourcen-Handle verwendet. Andernfalls ist das Datenelement auf FALSE gesetzt.

Mode

```
uint16 Mode;
```

Wenn *Mode* den Wert *DIB_RGB_Colors* hat, enthält die Farbtabelle 4-Byte große RGB-Einträge. Wenn *Mode* den Wert *DIB_PAL_COLORS* hat, enthält die Farbtabelle 2-Byte große Indizes in eine andere Palette (wie zum Beispiel die Systempalette). Da immer einer der beiden Fälle existieren müßte, sind bei bestimmten Funktionen zwei Versionen (wie zum Beispiel *GetColors()* und *GetIndices()*) erforderlich.

Siehe auch: *TDib::GetColors*, *TDib::GetIndices*, *TDib::Usage*

NumClrs

```
long NumClrs;
```

Die Zahl der mit dieser DIB assoziierten Farben.

Siehe auch: *TDib::NumColors*

W

```
int W;
```

Die Breite der DIB in Pixel.

Siehe auch: *TDib::Width*, *TDib::Size*

Geschützte Elementfunktionen

InfoFromHandle

```
void InfoFromHandle();
```

Sperrt das Handle dieser DIB und extrahiert die verbleibenden Datenelementwerte aus dem DIB-Header.

Siehe auch: *TDib::GetInfoHeader*

LoadFile

```
bool LoadFile(const char* name);
```

Lädt diese DIB vom gegebenen Dateinamen. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDib::LoadResource*, *TDib* Konstruktoren

LoadResource

```
bool LoadResource(HINSTANCE instance, TResID resID);
```

Lädt diese DIB von der gegebenen Ressource und gibt bei Erfolg TRUE zurück.

Siehe auch: *TDib::LoadFile*, *TDib* Konstruktoren

Read

```
bool Read(TFile& file, long offBits = 0);
```

Liest Daten für diese DIB von einer beliebigen Datei, BMP oder Ressource. Die Daten starten bei Offset *offBits*. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TDib::LoadFile*

Klasse TDibDC — dc.h

Die von *TDC* abgeleitete DC-Klasse *TDibDC* stellt den Zugriff auf geräteunabhängige Bitmaps (DIBs = Device-Independent Bitmaps) bereit.

Öffentliche Konstruktoren

Konstruktor

```
TDibDC(const TDib& dib);
```

Erzeugt ein *TDibDC*-Objekt mit den durch das gegebene *TDib*-Objekt bereitgestellten Daten.

Siehe auch: *class TDib*, *TDC::TDC*

Klasse TDocument — docview.h

TDocument ist eine abstrakte Basisklasse, die als eine Schnittstelle zwischen dem Dokument, seinen Views und der Dokumentverwaltung (Klasse *TDocManager*) dient. *TDocument* erzeugt, entfernt und sendet Botschaften über die View. Wenn der Benutzer zum Beispiel das Dokument ändert, teilt *TDocument* der View mit, daß das Dokument aktuali-

siert wurde. Das Makro DEFINE_DOC_TEMPLATE_CLASS verknüpft ein Dokument mit seinen Views.

Um Botschaften an seine assoziierten Views zu schicken, verwaltet das Dokument eine Liste aller für das Dokument existierender Views und kommuniziert mit den Views über den ObjectWindows-Ereignisbehandlungs-Mechanismus. Anstatt die Funktion *SendMessage* zu benutzen, greift das Dokument auf die Ereignistabelle der View zu. Die Views aktualisieren die Daten des Dokumentes über einen Aufruf die Elementfunktionen des speziellen Dokumentes. Views können ferner Streams anfordern, die durch das Dokument konstruiert werden.

Sowohl Dokumente als auch Views müssen Listen mit Eigenschaften für deren Anwendungen benutzen. Wenn Dokumente und Views erzeugt oder entfernt werden, werden Botschaften an die Anwendung geschickt. Diese kann dann die Eigenschaften abfragen, um zu bestimmen, wie das Dokument oder die View zu verarbeiten ist. Es liegt in der Verantwortung der Dokumentverwaltung, zu bestimmen, ob eine spezielle View passend für das gegebene Dokument ist.

Da die Funktionen der Eigenschaftsattribute virtuell sind, müßte eine abgeleitete Klasse (die zuerst aufgerufen wird) die in einer Basisklasse definierten Eigenschaften überschreiben. Jede abgeleitete Klasse muß ihre eigenen Eigenschaftsattribut-Typen entweder als String oder als binäre Daten implementieren. Wenn die abgeleitete Klasse die Eigenschaftsnamen der übergeordneten Klasse verdoppelt, sollte sie dasselbe Verhalten und denselben Datentyp wie in der übergeordneten Klasse bereitstellen.

Um Dokumente persistenter zu machen, enthält *TDocument* diverse virtuelle Funktionen (beispielsweise *InStream* und *OutStream*), die Streaming unterstützen. Ihre abgeleiteten Klassen müssen diese Funktionen außer Kraft setzen, um Daten lesen und schreiben zu können.

Auch wenn Dokumente üblicherweise mit Dateien assoziiert sind, müssen sie nicht notwendigerweise Dateien sein. Sie können ebenso aus Datenbanken, Tabellen, Briefsystemen, Fax- oder Modem-Übermittlungen, Plattenverzeichnissen usw. bestehen.

Öffentliche Datenelemente

ChildDoc

```
List ChildDoc;
```

Die Liste der mit diesem Dokument assoziierten, untergeordneten Dokumente.

Tag

```
void far* Tag;
```

Tag enthält einen Zeiger auf die durch die Anwendung definierten Daten. Normalerweise können Sie *Tag* verwenden, um einen Zeiger auf Ihre eige-

ne assoziierte Datenstruktur der Anwendung zu installieren. *Tag* wird zum dem Zeitpunkt, wenn ein *TDocument* konstruiert wird, mit Null initialisiert und wird sonst von den Dokument-View-Klassen nicht benutzt.

Typdefinition

TDocProp

```
enum TDocProp{PrevProperty = 0, DocumentClass, TemplateName, ViewCount,
StoragePath, DocTitle, NextProperty};
```

Diese Eigenschaftswerte sind für *TDocument* definiert und sind in Klassen verfügbar, die von *TDocument* abgeleitet sind. *PrevProperty* und *NextProperty* sind Begrenzer für jede Eigenschaftsliste des Dokuments.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TDocument(TDocument* parent = 0);
```

Obwohl Sie ein *TDocument*-Objekt nicht direkt erzeugen, müssen Sie den Konstruktor aufrufen, wenn Sie eine abgeleitete Klasse erzeugen. *parent* zeigt auf das zum neuen Dokument übergeordnete Dokument. Wenn kein übergeordnetes Dokument existiert, ist *parent* Null.

Destruktor

```
virtual ~TDocument();
```

Löscht ein *TDocument*-Objekt. Normalerweise wird zuerst *Close* aufgerufen. *~TDocument* löscht alle untergeordneten Dokumente und schließt alle offenen Streams. Wenn dies das letzte Dokument ist, welches das Template benutzt, schließt die Funktion das Template des Objektes und alle assoziierten Views, löscht den Stream des Objektes und entfernt sich selbst aus der im übergeordneten Dokument befindlichen Liste untergeordneter Dokumente (sofern ein übergeordnetes Dokument existiert). Wenn kein übergeordnetes Dokument existiert, entfernt die Funktion ihr Dokumentobjekt aus der Dokumentliste der Dokumentverwaltung.

Siehe auch: *TDocument::Close*

Öffentliche Elementfunktionen

CanClose

```
virtual bool CanClose();
```

Überprüft, ob alle untergeordneten Dokumente geschlossen werden können, bevor das aktuelle Dokument geschlossen wird. Wenn eines der untergeordneten Dokumente FALSE zurückliefert, gibt auch *CanClose* FALSE zurück und der bricht den Prozeß ab. Wenn alle untergeordneten Dokumente TRUE zurückgeben, ruft die Funktion *TDocManager::FlushDoc* auf. Wenn *FlushDoc* ermittelt, daß das Dokument verändert, aber nicht gespeichert wurde, zeigt die Funktion eine Meldung an, die den Benutzer fragt, das Dokument entweder zu speichern, alle Änderun-

gen zu verwerfen oder die Operation abzubrechen. Wenn das Dokument nicht verändert wurde, und die *CanClose*-Funktionen aller untergeordneten Dokumente TRUE zurückgeben, gibt auch diese *CanClose*-Funktion TRUE zurück.

Siehe auch: *TDocManager::FlushDoc*

Close

```
virtual bool Close();
```

Schließt das Dokument, löscht oder entfernt das Dokument allerdings nicht. Vor dem Schließen des Dokumentes überprüft *Close* alle untergeordneten Dokumente und versucht, diese zu schließen, bevor das übergeordnete Dokument geschlossen wird. Rufen Sie die *Close*-Version von *TDocument* auch auf, wenn Sie Ihre eigene *Close*-Funktion schreiben, damit alle untergeordneten Dokumente überprüft werden, bevor das übergeordnete Dokument geschlossen wird.

Commit

```
virtual bool Commit(bool force = false);
```

Sichert die aktuellen Daten in den Speicher. Wenn eine Datei geschlossen wird, ruft die Dokumentverwaltung entweder *Commit* oder *Revert* auf. Wenn *force* TRUE ist, werden alle Daten in den Speicher geschrieben. Die *Commit*-Funktion von *TDocument* überprüft alle untergeordneten Dokumente und übergibt auch deren Änderungen an den Speicher. Bevor die aktuellen Daten gespeichert werden, müssen alle untergeordneten Dokumente TRUE zurückgeben. Wenn alle untergeordneten Dokumente TRUE zurückgeben, leert *Commit* die Views für Operationen, die seit der letzten Überprüfung der View vorkamen. Sobald alle Daten für das Dokument aktualisiert und gesichert sind, gibt *Commit* TRUE zurück.

Siehe auch: *TDocument::Revert*

FindProperty

```
virtual int FindProperty(const char far* name);
```

FindProperty ermittelt über den gegebenen Eigenschaftsnamen (*name*) den Eigenschaftsindex. Gibt einen Ganzzahlindex zurück, der zum Namen korrespondiert, oder Null, wenn der Name in der Eigenschaftsliste nicht gefunden wurde.

Siehe auch: *pfxxxx Eigenschaftsattribut*-Konstanten, *TDocument::PropertyName*

GetDocManager

```
TDocManager& GetDocManager();
```

Gibt einen Zeiger an die aktuelle Dokumentverwaltung zurück.

GetDocPath

```
const char far* GetDocPath();
```

Gibt den Verzeichnispfad für das Dokument zurück. Dies könnte die Speichern-unter-Operation ändern.

GetOpenMode

```
int GetOpenMode;
```

Liefert die Modus- und Schutz-Flag-Werte für das aktuelle Dokument.

Siehe auch: *TDocument::SetOpenMode*

GetParentDoc

```
TDocument* GetParentDoc():
```

Gibt das zum aktuellen Dokument übergeordnete Dokument oder Null, wenn es kein übergeordnetes Dokument gibt, zurück.

GetProperty

```
virtual int GetProperty(int index, void far* dest, int textlen=0);
```

Gibt die Gesamtzahl von Eigenschaften für dieses Dokument zurück, wobei *index* der Eigenschaftsindex ist, *dest* die Eigenschaftsdaten enthält und *textlen* die Größe des Array ist. Wenn *textlen* Null ist, werden die Eigenschaftsdaten als binäre Daten zurückgegeben. Andernfalls werden die Eigenschaftsdaten als Textdaten zurückgegeben.

Siehe auch: *pfxxxx Eigenschaftsattribut*-Konstanten,
TDocument::SetProperty

GetTemplate

```
TDocTemplate* GetTemplate();
```

Liefert das Template, das für die Dokumenterzeugung benutzt wird. Das Template kann während einer Speichern-unter-Operation verändert werden.

GetTitle

```
CONST CHAR FAR GetTitle();
```

Gibt den Dokumenttitel zurück.

HasFocus

```
bool HasFocus(HWND hwnd);
```

Wird von der Dokumentverwaltung benutzt. *HasFocus* gibt TRUE zurück, wenn die View dieses Dokuments den Fokus besitzt. *hwnd* ist ein Handle auf das Dokument, mit dem bestimmt werden kann, ob das Dokument eine View mit einem Fokus enthält.

Die mit diesem Dokument assoziierte View ist die aktive View.

InitDoc

```
virtual bool InitDoc();
```

Die *TDocument*-Elementfunktion *InitDoc* stellt eine virtuelle Methode dar, die von der *TOleDocument*-Elementfunktion *InitDoc* außer Kraft gesetzt wird. Sie können diese Funktion verwenden, um das Dokument vor dem Ereignis *dnCreate* vorzubereiten. Dieses Ereignis bedeutet, daß das Dokument vor der Erstellung der View erzeugt und eingetragen wurde.

Siehe auch *TOleDocument::InitDoc*, Aufzählungstyp Dokumentbotschaft *dnxxxx*

InStream

```
virtual TInStream* InStream(int mode, LPCSTR strmId=0);
```

Generische Eingabe für das spezielle Speichermedium. *InStream* gibt einen Zeiger auf ein *TInStream* zurück. *mode* ist eine Kombination der ios-Bits, die in iostream.h definiert sind. *strmId* ist eine der *shxxxx*-Datei-

zugriffsmodi. *strmId* wird für Dokumente benutzt, die benannte Streams unterstützen. Überschreiben Sie diese Funktion, um Streaming für Ihre Dokumentklasse bereitzustellen.

Siehe auch: *TDocument::OutStream*

IsDirty

```
virtual bool IsDirty();
```

Gibt TRUE zurück, wenn das Dokument oder eine seiner Views verändert, aber nicht gesichert wurde.

IsEmbedded

```
virtual bool IsEmbedded();
```

Gibt TRUE zurück, wenn das Dokument in einen OLE2-Container eigebettet ist.

Siehe auch *TDocument::SetEmbedded*

IsOpen

```
virtual bool IsOpen();
```

Überprüft, ob das Dokument irgendwelche Streams in seiner Stream-Liste besitzt. Gibt FALSE zurück, wenn keine Streams offen sind. Andernfalls gibt die Funktion TRUE zurück.

NextStream

```
TStream* NextStream(const TStream* strm);
```

Liefert den nächsten Eintrag in der Stream-Liste. Enthält Null, wenn keiner existiert.

NextView

```
TView* NextView(const TView* view);
```

Liefert die nächste View in der View-Liste. Enthält Null, wenn keine existiert.

NotifyViews

```
bool NotifyViews(int event, long item=0, TView* exclude=0);
```

Benachrichtigt die Views des aktuellen Dokumentes und die Views aller untergeordneten Dokumente über eine Änderung. Im Gegensatz zu *QueryViews* sendet *NotifyViews* eine Benachrichtigung über ein Ereignis an alle Views und gibt TRUE zurück, wenn irgendwelche Views TRUE als Ergebnis zurückgegeben haben. Das Ereignis EV_OWLNOTIFY wird mit einem Ereignis-Code, der für das spezielle Dokument und View privat ist, sowie mit einem **long**-Argument gesendet, welches in den aktuellen Typ passend umgewandelt werden kann, der im Argument der Antwortfunktion übergeben wird.

Siehe auch: *TDocument::QueryViews*

Open

```
virtual bool Open(int mode, const char far* path = 0);
```

Öffnet das Dokument mit Hilfe des durch *DocPath* angegebenen Pfades. Setzt *OpenMode* auf *mode*. Die Version von *TDocument* gibt immer TRUE zurück und führt sonst keine weiteren Aktionen durch. Andere Klassen

überschreiben diese Funktion, um die angegebenen Dokumente und Views für Dateien zu öffnen.

Siehe auch: *TFileDocument::Open*

OutStream

```
virtual TOutStream* OutStream(int mode, const char far* strmId = 0);
```

Generische Ausgabe für ein spezielles Speichermedium. *OutStream* gibt einen Zeiger auf ein *TOutStream* zurück. *mode* ist eine Kombination der ios-Bits, die in iostream.h definiert sind. *strmId* wird für Dokumente benutzt, die benannte Streams unterstützen. Die Version von *TDocument* gibt immer Null zurück. Überschreiben Sie diese Funktion, um Streaming für Ihre Dokumentklasse bereitzustellen.

Siehe auch: *TDocument::InStream*

PostError

```
uint PostError(uint sid, uint choice = MB_OK);
```

Hinterlegt eine Fehlermeldung, die in *sid* als String-Ressoucen-ID übergeben wird. *choice* ist eine oder mehrere der MB_Xxxx-Eigenschaftskonstanten.

Siehe auch: *TDocManager::PostDocError*, *MB_Xxxx* Botschaftskonstanten

PropertyCount

```
virtual int PropertyCount();
```

Liefert die Gesamtzahl von Eigenschaften für das *TDocument*-Objekt. Gibt *NextProperty* -1 zurück.

Siehe auch: *pfxxxx* Eigenschaftsattribut-Konstanten

PropertyFlags

```
virtual int PropertyFlags(int index);
```

Gibt die Attribute der angegebenen Eigenschaft zurück. Dabei ist der Index (*index*) der Eigenschaft gegeben, deren Attribute Sie ermitteln wollen.

Siehe auch: *pfxxxx Eigenschaftsattribut*-Konstanten, *TDocument::FindProperty*, *TDocument::PropertyName*

PropertyName

```
virtual const char* PropertyName(int index);
```

Gibt den Namen der Eigenschaft bei gegebenem Indexwert (*index*) zurück.

Siehe auch: *pfxxxx Eigenschaftsattribut*-Konstanten, *TDocument::FindProperty*

QueryViews

```
TView* QueryViews(int event, long item=0, TView* exclude=0);
```

Fragt die Views des aktuellen Dokumentes und die Views aller untergeordneten Dokumente über das angegebene Ereignis ab, stoppt jedoch bei der ersten View, die TRUE zurückgibt. Im Gegensatz zu *NotifyViews* gibt *QueryViews* einen Zeiger auf die erste View zurück, die auf ein Ereignis mit TRUE antwortet. Das Ereignis EV_OWLNOTIFY wird mit einem Ereignis-Code (der für die spezielle Dokument- und View-Klasse privat ist) sowie mit einem **long**-Argument (welches in den aktuellen Typ, der im

Argument der Antwortfunktion übergeben wird, passend umgewandelt werden kann) gesendet.

Siehe auch: *TDocument::NotifyViews*

Revert

```
virtual bool Revert( bool clear = false);
```

Führt das Gegenteil von *Commit* aus und verwirft alle Änderungen, die an dem Dokument seit dem letzten *commit* vorgenommen wurden. Wenn *clear* TRUE ist, werden die Daten für Views nicht neu geladen. *Revert* überprüft zudem alle untergeordneten Dokumente und verwirft alle Änderungen, wenn alle untergeordneten Dokumente TRUE zurückgeben. Wenn eine Datei geschlossen wird, ruft die Dokumentverwaltung *Commit* oder *Revert* auf. Gibt TRUE zurück, wenn die Aufruf erfolgreich ist.

Siehe auch: *TDocument::Commit*

RootDocument

```
virtual TDocument& RootDocument();
```

Gibt den **this**-Zeiger als das Stammdokument zurück.

SetDocManager

```
void SetDocManager(TDocManager& dm);
```

Setzt die aktuelle Dokumentverwaltung auf das Argument *dm*.

SetDocPath

```
virtual bool SetDocPath(const char far* path);
```

Setzt den Dokumentpfad für Öffnen- und Speichern-Operationen.

SetEmbedded

```
virtual bool SetEmbedded();
```

SetEmbedded markiert das Dokument als in einen OLE2-Container eingebettet. Dies geschieht normalerweise bei der Erzeugung des Servers und wenn das Factory-Klassen-Template die Komponente erzeugt.

Siehe auch *TDocument::IsEmbedded*

SetOpenMode

```
void SetOpenMode(int mode);
```

Setzt die Modus- und Schutz-Flag-Werte für das aktuelle Dokument.

Siehe auch: *TDocument::GetOpenMode*

SetProperty

```
virtual bool SetProperty(int index, const void far* src);
```

Setzt den Wert der Eigenschaft bei gegebenem Eigenschaftsindex und *src*, dem Datentyp (entweder binär oder Text), auf den die Eigenschaft gesetzt werden muß.

Siehe auch: *pfxxxx* Eigenschaftsattribut-Konstanten, *TDocument::GetProperty*

SetTemplate

```
bool SetTemplate(TDocTemplate* tpl);
```

Setzt das Dokument-Template. Wenn allerdings der Template-Typ mit der Datei inkompatibel ist, läßt die Dokumentverwaltung nicht zu, daß die Datei unter diesem Template-Typen gespeichert wird.

SetTitle

```
virtual void SetTitle(const char far* title);
```

Setzt den Titel des Dokumentes.

Geschützte Datenelemente

DirtyFlag

```
bool DirtyFlag;
```

Zeigt an, daß ungesicherte Änderungen am Dokument vorgenommen wurden. Views können auch unabhängig ihren lokalen Disketten- und Plattenstatus verwalten.

Embedded

```
bool Embedded;
```

Gibt an, ob das Dokument eingebettet ist.

Geschützte Elementfunktionen

AttachStream

```
virtual void AttachStream(TStream& strm);
```

AttachStream wird vom *TStream*-Konstruktor aufgerufen und verbindet einen Stream mit dem aktuellen Dokument.

DetachStream

```
virtual void DetachStream(TStream& strm);
```

DetachStream wird vom *TStream*-Destruktor aufgerufen und löst die Verbindung eines Stream mit dem aktuellen Dokument.

Eingebettete Klasse TDocument::List — docview.h

Die eingebettete Klasse *TDocument::List* kapselt die Kette mit Dokumenten. Sie erlaubt das Hinzufügen und Entfernen von Dokumenten aus der Dokumentliste.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
List();
```

Konstruiert ein *TDocument::List*-Objekt.

Destruktor

```
~List();
```

Entfernt ein *TDocument::List*-Objekt.

Öffentliche Elementfunktionen

Destroy

```
void Destroy();
```

Löscht alle Dokumente.

Insert

```
bool Insert(TDocument* doc);
```

Fügt ein neues Dokument in die Dokumentliste ein. Schlägt fehl, wenn des Dokument bereits existiert.

Next

```
TDocument* Next(const TDocument* doc);
```

Next gibt das erste Dokument der Dokumentliste zurück, wenn der Parameter *doc* 0 ist.

Remove

```
bool Remove(TDocument* doc);
```

Entfernt ein Dokument aus der Dokumentliste.

Klasse TDocManager — docmanag.h

TDocManager erzeugt ein Dokumentverwaltungsobjekt, das die Liste aktueller Dokumente und registrierter Templates verwaltet, Standardkommandos des Dabeimenüs behandelt und die Benutzeroberfläche für Datei- und View-Auswahlfenster anzeigt. Um eine Unterstützung für Dokumente und Views bereitzustellen, muß eine Instanz von *TDocManager* durch die Anwendung erzeugt und mit der Anwendung verbunden werden.

Die Dokumentverwaltung behandelt normalerweise Ereignisse im Namen der Dokumente, indem sie eine Antworttabelle benutzt, um die Standardkommandos CM_FILENEW, CM_FILEOPEN, CM_FILECLOSE, CM_FILESAVE, CM_FILESAVEAS, CM_FILEREVERT, CM_FILEPRINT, CM_FILEPRINTERSETUP und CMVIEWCREATE des *Datei*-Menüs zu verarbeiten. Als Antwort auf ein CM_FILENEW- oder ein CM_FILEOPEN-Kommando erzeugt die Dokumentverwaltung basierend auf der Auswahl des Benutzers das passende Dokument. Als Antwort auf die anderen Kommandos bestimmt die Dokumentverwaltung, welches offene Dokument die View beinhaltet, die mit dem fokussierten Fenster assoziiert ist. Die Menükommandos werden zuerst zu dem aktuell fokussierten Fenster, dann über die Kette übergeordneter Fenster an das Hauptfenster und schließlich an die Anwendung geschickt, die die Kommandos an die Dokumentverwaltung weiterleitet.

Wenn Sie eine *TDocManager* oder eine davon abgeleitete Klasse erzeugen, müssen Sie festlegen, ob sie entweder ein Mehrdokumentoberfläche (*dmMDI*) oder Einzeldokumentoberfläche (*dmSDI*) besitzt. Wenn die Dokumentverwaltung zusätzlich die Standarddateikommandos behandeln soll, müssen Sie diese mit *dmMenu* »ver-ODER-n«.

Sie können zudem die Menüoptionen der Dokumentverwaltung aktivieren oder deaktivieren, indem Sie *dmSaveEnable* oder *dmNoRevert* im Konstruktor übergeben. Wenn Sie die Menüoption *Datei/Speichern* aktivieren wollen, bei unverändertem Dokument, dann übergeben Sie das Flag *dmSaveEnable* im Konstruktor. Zum Deaktivieren der Menüoption für das Wiedereinspielen einer gesicherten Version übergeben Sie *dmNoRevert* im Konstruktor.

Wenn die Anwendung ein neues Dokument und eine neue View direkt erzeugt, kann sie die View an ihre Rahmenfenster anschließen, ein untergeordnetes MDI-Fenster erzeugen (MDI = Multiple Document Interface, Multidokumentschnittstelle), das Fenster in Gang bringen oder einen Splitter erzeugen (Aufspalten des Fensters). Wenn allerdings die Dokumentverwaltung ein neues Dokument und eine neue View von der Menüauswahl *Datei/Öffnen* oder *Datei/Neu* erzeugt, wird die Anwendung den Prozeß nicht kontrollieren. Damit die Anwendung die Kontrolle erhält, sendet die Dokumentverwaltung Botschaften nach der erfolgreichen Erzeugung des neuen Dokumentes und der View. Dann kann die Anwendung die im Template enthaltene Information benutzen, um zu bestimmen, wie das neue Dokument- und View-Objekt zu installieren ist.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TDocManager(int mode, TDocTemplate*& templateHead =
DocTemplateStaticHead);
```

Konstruiert ein *TDocManager*-Objekt, das abhängig von der Anwendung entweder einzelne (SDI = Single Document Interface, Einzeldokumentschnittstelle) oder mehrere (MDI = Multiple Document Interface, Mehrdokumentschnittstelle) offene Dokumente unterstützt. *mode* ist entweder auf *dmMenu*, *dmMDI*, *dmSDI*, *dmSaveEnable* oder *dmNoRevert* gesetzt. Um die Standardkommandos des *Datei*-Menüs von *TDocManager* zu installieren, müssen Sie *dmMDI* oder *dmSDI* mit *dmMenu* »ver-ODER-n«. Ein Beispiel:

```
DocManager = new TDocManager(DocMode | dmMenu);
```

Die Dokumentverwaltung kann dann ihr Menü und ihre Antworttabelle verwenden, um diese Ereignisse zu behandeln. Wenn Sie den Parameter *dmMenu* nicht angeben, müssen Sie das Menü wie auch die Funktionen zur Behandlung dieser Kommandos bereitstellen. Sie können aber weiterhin das Datenelement *DocManager* Ihres Anwendungsobjektes verwenden, um auf die Funktionen der Dokumentverwaltung zuzugreifen.

Konstruktor

```
TDocManager(int mode, TApplication* app, TDocTemplate*& templateHead =
DocTemplateStaticHead);
```

Der Konstruktor führt die gleichen Operationen durch wie der erste Konstruktor. Der zusätzliche Parameter *app* zeigt jedoch auf die mit diesem Dokument assoziierte Anwendung.

Destruktor

```
virtual ~TDocManager();
```

Löscht ein *TDocManager*-Objekt und entfernt alle verbundenen Dokumente und Templates. Der Konstruktor setzt *TDocTemplate::StaticHead* zurück, so daß es auf den Kopf der statischen Template-Liste zeigt.

Siehe auch: *dmxxxx* Moduskonstanten der Dokumentverwaltung

Öffentliche Datenelemente

DocList

```
TDocument::List DocList;
```

Enthält die Liste der angeschlossenen Dokumente oder Null, wenn keine Dokumente existieren.

Öffentliche Elementfunktionen

AttachTemplate

```
void AttachTemplate(TDocTemplate&);
```

Fügt ein Template in die Template-Kette ein.

CmFileClose

```
virtual void CmFileClose();
```

Antwortet auf eine Botschaft zum Schließen einer Datei. Überprüft, ob das Dokument seit der letzten Sicherung verändert wurde und verlangt eventuell vom Benutzer eine Bestätigung der Speicheroperation.

CmFileNew

```
virtual void CmFileNew();
```

Ruft *CreateAnyDoc* ohne Pfadangabe auf.

Siehe auch: *TDocManager::CreateAnyDoc*, *dtxxxx* Dokument-Template-Konstanten

CmFileOpen

```
virtual void CmFileOpen();
```

Ermöglicht dem Benutzer die Auswahl einer registrierten Template aus der im Dialogfenster angezeigten Liste. Ruft *CreateAnyDoc* auf.

Siehe auch: *TDocManager::CreateAnyDoc*

CmFileRevert

```
virtual void CmFileRevert();
```

Kehrt zum vorher gesicherten Dokument zurück. Kehrt nicht zum Dokument zurück, wenn dieses seit der letzten Sicherung nicht verändert wurde, d. h., wenn die Funktion *IsDirty* des Dokuments FALSE zurückgibt.

CmFileSave

```
virtual void CmFileSave();
```

Antwortet auf eine Botschaft zum Speichern der Datei. Setzt *doc* auf das aktuelle Dokument. *CmFileSave* prüft *IsDirty* nur dann, wenn das Flag dmSaveEnable nicht angegeben wurde. Ruft *PostDocError* mit IDS_NOTCHANGED auf, wenn *dmSaveEnable* NICHT angegeben wurde und *IsDirty* FALSE zurückgibt.

Siehe auch IDS_xxxx Dokument-String-ID-Konstanten

CmFileSaveAs

```
virtual void CmFileSaveAs();
```

Fordert den Benutzer auf, einen neuen Namen für das Dokument einzugeben und speichert das Dokument in dieser Datei.

CmViewCreate

```
virtual void CmViewCreate();
```

Erzeugt eine Dokument-View auf Basis des View-Namens des aktuellen Dokuments. Gibt es mehr als ein Template für das Dokument, erlaubt es *CmViewCreate* dem Anwender, den View-Typ aus der Template-Liste auszuwählen.

CreateAnyDoc

```
virtual TDocument* CreateAnyDoc(const char far* path, long flags= 0);
```

Erzeugt ein Dokument, das auf dem Verzeichnispfad und dem angegebenen Template basiert. *flags*, eine der Dokument-Template-Konstanten, legt fest, wie das Dokument-Template erzeugt wird. Wenn *path* Null ist und es sich nicht um ein neues Dokument handelt (das Flag *dtNewDoc* ist nicht gesetzt), wird ein Dialogfenster angezeigt. Wenn *path* Null ist, *dtNewDoc* nicht gesetzt ist und mehr als ein Template existiert, werden ein Dialogfenster und eine Liste mit Templates angezeigt.

Siehe auch: *TDocTemplate::CreateDoc*. *dtxxxx* Dokument-Template-Konstanten

CreateAnyView

```
virtual TView* CreateAnyView(TDocument& doc, long flags= 0);
```

Erzeugt eine Dokument-View, die auf dem Verzeichnispfad und dem angegebenen Template basiert. *flags*, eine der Dokument-Template-Konstanten, legt fest, wie das Dokument-Template erzeugt wird.

Siehe auch: *TDocTemplate::CreateView*, *dtxxxx* Dokument-Template-Konstanten

CreateDoc

```
TDocument* CreateDoc(TDocTemplate* tpl, const char far*, TDocument*
parent, long flags = 0);
```

CreateDoc erzeugt basierend auf dem Verzeichnispfad und dem angegebenen Template ein Dokument. *flags* enthält eine der Dokument-Template-Konstanten, die festlegt, wie das Dokument erzeugt wird.

Siehe auch Klasse *TDocument, TDocTemplate::CreateView, dtxxxx* Dokument-Template-Konstanten

CreateView

```
TView* CreateView(TDocument* doc);
```

CreateView erzeugt eine View des angegebenen Dokuments.

Siehe auch *TDocTemplate::CreateView, dtxxxx* Dokument-Template-Konstanten

DeleteTemplate

```
void DeleteTemplate(TDocTemplate&);
```

Entfernt ein Template aus der mit dem Dokument verbundenen Liste von Templates.

Siehe auch: *TDocManager::RefTemplate*

EvCanClose

```
bool EvCanClose();
```

Überprüft, ob alle untergeordneten Dokumente geschlossen werden können, bevor das aktuelle Dokument geschlossen wird. Wenn eines der untergeordneten Dokumente FALSE zurückgibt, gibt auch *EvCanClose* FALSE zurück und bricht den Prozeß ab. Wenn alle untergeordneten Dokumente TRUE zurückgeben, ruft *EvCanClose* die Funktion *TDocManager::FlushDoc* für jedes Dokument auf. Wenn *FlushDoc* das Dokument als *dirty* erkennt, zeigt die Funktion eine Meldung an, die vom Benutzer eine Entscheidung darüber fordert, ob das Dokument zu speichern ist, ob die Änderungen verworfen werden sollen oder ob die Operation abzubrechen ist. Wenn das Dokument nicht *dirty* ist und *EvCanClose* TRUE zurückgibt, gibt auch *EvCanClose* TRUE zurück.

Siehe auch: *TApplication::CanClose*, *TDocManager::FlushDoc*

EvPreProcessMenu

```
void EvPreProcessMenu(HMENU hmenu);
```

Wenn *EvPreProcessMenu* von *MainWindow* aus aufgerufen wird, lädt und löscht die Funktion ein Menü an der durch MF_POSITION oder MF_POPUP angegebenen Position. Ihre Anwendung kann *EvPreProcessMenu* aufrufen, um das Menü des Hauptfensters zu verarbeiten, bevor es angezeigt wird.

Siehe auch: *TApplication::PreProcessMenu*

EvWakeUp

```
void EvWakeUp();
```

Wird nur nach dem Streaming in der Dokumentverwaltung verwendet. *EvWakeUp* ermöglicht die Erzeugung des Fensters nach dem Streaming.

FindDocument

```
TDocTemplate* FindDocument(const char far*path);
```

FindDocument gibt das erste Dokument zurück, dessen Maske dem gegebenen Dateinamen entspricht. Wird zu diesem Dateinamen kein Dokument gefunden, oder ist das Dokument bereits geöffnet, wird 0 zurückgegeben.

Siehe auch Klasse *TDocTemplate*

FlushDoc

```
virtual bool FlushDoc(TDocument& doc);
```

Aktualisiert das Dokument mit allen Änderungen und fordert vom Benutzer eine Bestätigung der Aktualisierungen.

GetApplication

```
TApplication* GetApplication();
```

Gibt die aktuelle Anwendung zurück.

GetCurrentDoc

```
virtual TDocument* GetCurrentDoc();
```

Ruft *TWindow::GetFocus* auf, um das Fenster mit dem Fokus zu bestimmen. Durchsucht die Liste mit Dokumenten und gibt das Dokument wieder, das die View mit dem Fokus enthält. Gibt Null zurück, wenn kein Dokument eine View mit Fokus hat.

Siehe auch Klasse *TApplication*

GetNextTemplate

```
TDocTemplate* GetNextTemplate(TdocTemplate* tpl);
```

Gibt das nächste Dokument-Template zurück.

Siehe auch Klasse *TDocTemplate*

InitDoc

```
TDocument* InitDoc(TDocument*doc, const char far* path, long flags);
```

Initialisiert die Dokumente, den Verzeichnispfad für das Dokument und die *dtXXXX*-Dokument-Flag-Werte (wie beispielsweise *dtNewDoc*), die zur Erzeugung von Dokument-Templates verwendet werden.

Siehe auch Klasse *TDocTemplate,* Dokument-View-Konstanten

IsFlagSet

```
bool IsFlagSet(int Flag);
```

Gibt TRUE zurück, wenn die *dtXXXX*-Dokument-Template-Konstante, die in *Flag* angegeben wird, gesetzt ist.

Siehe auch Dokument-View-Konstanten

MatchTemplate

```
TDocTemplate* MatchTemplate(const char far* path);
```

MatchTemplate gibt das erste registrierte Template zurück, dessen Maske mit dem gegebenen Dateinamen übereinstimmt. Wenn kein Template mit dem gelieferten Dateinamen kompatibel oder das Template bereits geöffnet ist, gibt die Funktion Null zurück.

Siehe auch Klasse *TDocTemplate*

PostDocError

```
virtual uint PostDocError(TDocument& doc, uint sid, uint choice = MB_OK);
```

Zeigt ein Meldungsfenster mit der Fehlermeldung an, die als String-Ressourcen-ID in *sid* übergeben wird. Standardmäßig enthält das Meldungsfenster entweder einen »OK«-Schalter oder ein Fragezeichensymbol. Wenn keine Fehlermeldung gefunden werden kann, zeigt *PostDocError* die Meldung »Meldung nicht gefunden« an. *choice* kann eine oder mehrere der MB_Xxxx Stilkonstanten sein. Diese Funktion kann überschrieben werden.

Siehe auch: *TDocument::PostError, MB_Xxxx* Botschaftskonstanten

PostEvent

```
virtual void PostEvent(int id, TDocument& doc);
```

Wenn sich das aktuelle Dokument ändert, ruft die Funktion *::SendMessage* auf und übergibt eine WM_OWLDOCUMENT-Botschaft, um eine Änderung im Status des Dokumentes anzuzeigen.

PostEvent

```
virtual void PostEvent(int id, TView& view);
```

Wenn sich die aktuelle View ändert, ruft die Funktion *::SendMessage* auf und übergibt eine WM_OWLVIEW-Botschaft, um eine Änderung im Status der View anzuzeigen.

Siehe auch Klasse *TDocument, TView*

RefTemplate

```
void RefTemplate(TDocTemplate&);
```

Fügt ein Template der mit dem Dokument verbundenen Liste mit Templates hinzu.

Siehe auch: *TDocManager::UnRefTemplate, TDocTemplate*

SelectAnySave

```
virtual TDocTemplate* SelectAnySave(TDocument& doc, bool samedoc = true);
```

Wählt ein registriertes Template aus, das mit dem Dokument zu speichern ist.

Siehe auch Klasse *TDocTemplate,* Klasse *TDocument*

SelectSave

```
bool SelectSave(TDocument& doc);
```

Fordert den Benutzer auf, einen Dateinamen für das Dokument auszusuchen. Filtert Dateien für reinen Lesezugriff heraus.

Siehe auch *TDocTemplate::SelectSave*

UnRefTemplate

```
void UnRefTemplate(TDocTemplate&);
```

Entfernt ein Template aus der mit dem Dokument verbundenen Liste mit Templates.

Siehe auch: *TDocManager::RefTemplate, TDocTemplate*

Geschützte Elementfunktionen

SelectDocPath

```
virtual int SelectDocPath(TDocTemplate** tpllist, int tplcount,
                          char far* path, int buflen, long flags,
                          bool save=false);
```

Fordert vom Benutzer die Auswahl eines der Templates, das für die zu öffnende Datei verwendet werden soll. Gibt den für die Auswahl verwendeten Template-Index oder Null bei nicht erfolgreichem Abschluß zurück. Für eine Operation zum Öffnen einer Datei ist *save* FALSE. Für eine Operation zum Speichern einer Datei ist *save* TRUE. Diese Funktion kann überschrieben werden, um eine selbstdefinierte Benutzeroberfläche bereitzustellen.

Siehe auch Klasse *TDocTemplate*

SelectDocType

```
virtual int SelectDocType(TDocTemplate** tpllist, int tplcount);
```

SelectDocType kann überschrieben werden. Die Funktion ermöglicht dem Benutzer die Auswahl eines Dokumenttypes aus einer Liste mit Dokument-Templates. Sie gibt den für die Auswahl verwendeten Template-Index oder Null bei nicht erfolgreichem Abschluß zurück.

Siehe auch Klasse *TDocTemplate*

SelectViewType

```
virtual int SelectViewType(TDocTemplate** tpllist, int tplcount);
```

SelectViewType kann überschrieben werden. Die Funktion ermöglicht dem Benutzer die Auswahl eines View-Namens für eine neue View aus einer Liste mit View-Namen. Sie gibt den für die Auswahl verwendeten Template-Index oder Null bei nicht erfolgreichem Abschluß zurück.

Siehe auch Klasse *TDocTemplate*

Anworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_COMMAND(CM_FILECLOSE, CmFileClose)	CmFileClose
EV_COMMAND(CM_FILENEW, CmFileNew),	CmFileNew
EV_COMMAND(CM_FILEOPEN, CmFileOpen)	CmFileOpen
EV_COMMAND(CM_FILEREVERT, CmFileRevert)	CmFileRevert
EV_COMMAND(CM_FILESAVE, CmFileSave)	CmFileSave
EV_COMMAND(CM_FILESAVEAS, CmFileSaveAs)	CmFileSaveAs
EV_COMMAND(CM_VIEWCREATE, CmViewCreate)	CmViewCreate
EV_COMMAND_ENABLE(CM_FILECLOSE, CmEnableClose)	CmEnableClose
EV_COMMAND_ENABLE(CM_FILENEW, CmEnableNew)	CmEnableNew
EV_COMMAND_ENABLE(CM_FILEOPEN, CmEnableOpen)	CmEnableOpen
EV_COMMAND_ENABLE(CM_FILEREVERT, CmEnableRevert)	EmEnableRevert
EV_COMMAND_ENABLE(CM_FILESAVE, CmEnableSave)	CmEnableSave
EV_COMMAND_ENABLE(CM_FILESAVEAS, CmEnableSaveAs)	CmEnableSaveAs
EV_COMMAND_ENABLE(CM_VIEWCREATE, CmEnableCreate)	CmEnableCreate
EV_WM_OWLCANCLOSE	EvCanClose
EV_WM_OWLPREPROCMENU	EvPreProcessMenu
EV_WM_WAKEUP	EvWakeUp

Struktur TEdgeConstraint — layoutco.h

TEdgeConstraint fügt Elementfunktionen hinzu, die Kantenbedingungen (aber nicht bezüglich der Größe) setzen. *TEdgeConstraint* plaziert Ihr Fenster immer ein Pixel über dem anderen Fenster und fügt dann Ränder hinzu.

Wenn der Rand zum Beispiel vier ist, plaziert *TEdgeConstraint* Ihr Fenster fünf Pixel über dem anderen Fenster. Der Rand, der nicht in Pixel gemessen werden muß, wird mit Hilfe der Einheiten definiert, die durch die Bedingung angegeben sind. Wenn deshalb der Rand in acht Layout-Einheiten angegeben ist (die dann zu zwölf Pixel konvertiert werden), würde Ihr Fenster dreizehn Pixel oberhalb des anderen Fensters plaziert werden. Siehe *TLayoutWindow* für ein Beispiel zu Layout-Bedingungen.

Siehe auch: *TLayoutConstraint*

Öffentliche Elementfunktionen

Above

```
inline void Above (TWindow *sibling, int margin = 0);
```

Positioniert Ihr Fenster über einem Schutzfenster. Sie müssen das Schutzfenster und einen optionalen Rand zwischen den zwei Fenstern angeben. Wenn kein Rand angegeben wird, setzt *Above* den Boden des Fensters einen Pixel über den Kopf des anderen Fensters.

Siehe auch: *TEdgeConstraint::Below*

Absolute

```
inline void Absolute (TEdge edge, int value);
```

Setzt eine Kante Ihres Fensters auf einen festen Wert.

Siehe auch: *TEdgeConstraint::PercentOf*

Below

```
inline void Below(TWindow *sibling, int margin = 0);
```

Positioniert Ihr Fenster in Abhängigkeit eines Schutzfensters. Sie müssen das Schutzfenster und einen optionalen Rand zwischen den zwei Fenstern angeben. Wenn kein Rand angegeben wird, setzt *Below* den Kopf des Fensters einen Pixel unter den Boden des anderen Fensters.

Siehe auch: *TEdgeConstraint::Above*

LeftOf

```
inline void LeftOf (TWindow *sibling;int margin = 0);
```

Positioniert Ihr Fenster in Abhängigkeit eines Schutzfensters. Sie können das Schutzfenster und einen optionalen Rand zwischen den zwei Fenstern angeben.

Siehe auch: *TEdgeConstraint::RightOf*

PercentOf

```
inline void PercentOf (TWindow *otherWin, TEdge edge, int percent);
```

Gibt an, daß die in *edge* angezeigte Kante des einen Fensters ein Teil der korrespondierenden Kante eines anderen Fensters oder *otherWin* sein sollte.

RightOf

```
inline void RightOf (TWindow *sibling, int margin = 0);
```

Positioniert Ihr Fenster in Abhängigkeit eines Schutzfensters. Sie können das Schutzfenster und einen optionalen Rand zwischen den zwei Fenstern angeben.

Siehe auch: *TEdgeConstraint::LeftOf*

SameAs

```
inline void SameAs (TWindow *otherWin, TEdge edge);
```

Setzt die durch *edge* angezeigte Kante Ihres Fensters äquivalent mit der korrespondierenden Kante des Fensters in *otherWin*.

Siehe auch: *TEdgeConstraint::Set*

Set

```
inline void Set(TEdge edge, TRelationship rel, TWindow *otherWin,
                TEdge otherEdge, int value = 0);
```

Wird für das Einstellen beliebiger Kantenbedingungen benutzt. *Set* gibt an, daß die Kante Ihres Fensters im angegebenen Verhältnis zu einer angegebenen Kante von *otherWin* stehen sollte.

Siehe auch: *TEdgeConstraint::SameAs*

Struktur TEdgeOrSizeConstraint — layoutco.h

TEdgeOrSizeConstraint ist von *TEdgeConstraint* abgeleitet und ist eine Template-Klasse, die Größenbedingungen zusätzlich zu allen von *TEdgeConstraint* bereitgestellten Operationen unterstützt. Die Breite und Höhe wird in der Template-Instanziierung dieser Klasse angegeben. Jede Elementfunktion besitzt zwei Versionen: die eine setzt sowohl Kanten- als auch Größenbedingungen; die andere setzt nur Kantenbedingungen.

Öffentliche Elementfunktionen

Absolute

```
inline void Absolute (int value);
```

Setzt die Breite und Höhe Ihres Fensters auf einen festen Wert.

Absolute

```
inline void Absolute (TEdge edge, int value);
```

Wird benutzt, um nur die Kantenbedingungen zu bestimmen. *Absolute* setzt die Kante Ihres Fensters auf einen festen Wert.

Siehe auch: *TEdgeConstraint::Absolute*

PercentOf

```
inline void PercentOf (TWindow *otherWin, int percent,
                       TWidthHeight otherWidthHeight = widthOrHeight);
```

Obwohl die Breite und Höhe eines Fensters standardmäßig ein Teil der korrespondierenden Dimensionen des Schutzfensters oder des übergeordneten Fensters sind, können sie ebenso ein Teil der entgegengesetzten Dimensionen des Schutzfensters oder des übergeordneten Fensters sein. So kann zum Beispiel die Breite eines Fensters 50% der Höhe des übergeordneten Fensters sein.

Siehe auch: *TEdgeOrSizeConstraint::Absolute*

PercentOf

```
inline void PercentOf (TWindow *otherWin, TEdge edge, int percent);
```

Wird benutzt, um nur die Kantenbedingungen zu bestimmen. *PercentOf* gibt an, daß die in *edge* angezeigte Kante des einen Fensters ein Teil der korrespondierenden Kante eines anderen Fensters oder *otherWin* sein sollte.

Siehe auch: *TEdgeConstraint::PercentOf*

SameAs

```
inline void SameAs (TWindow *otherWin,
                    TWidthHeight otherWidthHeight = widthOrHeight,
                    int value = 0);
```

Obwohl die Breite und Höhe eines Fensters standardmäßig ein Teil der korrespondierenden Dimensionen des Schutzfensters oder des übergeordneten Fensters sind, können sie ebenso gleich groß wie die entgegengesetzte Dimension des Schutzfensters oder des übergeordneten Fensters sein. So kann zum Beispiel die Breite eines Fensters mit der Höhe des übergeordneten Fensters übereinstimmen.

Siehe auch: *TEdgeOrSizeConstraint::PercentOf*

SameAs

```
inline void SameAs (TWindow *otherWin, TEdge edge);
```

Wird benutzt, um nur die Kantenbedingungen zu bestimmen. *SameAs* setzt die Kante des einen Fensters auf dieselben Angaben wie die korrespondierende Kante das in *otherWin* angegebenen anderen Fensters.

Siehe auch: *TEdgeConstraint::SameAs*

Klasse TEdit — edit.h

Ein *TEdit* ist ein Oberflächenobjekt, das ein Editierfeld als Oberflächenelement repräsentiert. Ein *TEdit*-Objekt muß für die Erzeugung eines Editierfeldes in einem übergeordneten *TWindow* verwendet werden. Ein *TEdit* kann zur Erleichterung der Kommunikation zwischen Ihrer Anwendung und den Editierfeldern eines *TDialog* benutzt werden. Diese Klasse ist streamfähig.

Für Editierfeldobjekte gibt es zwei Eigenschaften: einzeilig und mehrzeilig. Mehrzeilige Editierfelder erlauben die Bearbeitung mehrerer Textzeilen.

Die Position des ersten Zeichens in einem Editierfeld ist Null. Bei einem mehrzeiligen Editierfeld laufen die Positionsnummern von Zeile zu Zeile fort; Zeilenumbrüche zählen als zwei Zeichen.

De meisten Elementfunktionen von *TEdit* verwalten den Text des Editierfeldes. *TEdit* umfaßt zudem einige automatische Elementfunktionen, die auf Auswahlen im Menü des zum Editierfeld übergeordneten Fensters antworten. Dazu zählt auch das Ausschneiden, Kopieren und Einfügen. Zwei wichtige, von der Basisklasse von *TEdit* (*TStatic*) geerbte Elementfunktionen sind *GetText* und *SetText*.

Öffentliche Konstruktoren

Konstruktor

```
TEdit(TWindow* parent, int Id, const char far *text, int x, int y,
      int w, int h, uint textLen, bool multiline = false,
      TModule* module = 0);
```

Konstruiert ein Editierfeldobjekt mit einem übergeordneten Fenster (*parent*). Setzt die Erzeugungsattribute des Editierfeldes und füllt dessen *Attr*-Datenelemente mit der angegebenen Dialogelement-ID (*Id*), der Position (*x*, *y*) relativ zum Ursprung des Client-Bereiches des übergeordneten Fensters, der Breite (*w*) und der Höhe (*h*).

Wenn die Länge des Textpuffers (*textLen*) Null oder Eins ist, gibt es keine explizite Grenze für die Anzahl der Zeichen, die eingegeben werden können. Andernfalls können *textLen* - 1 Zeichen eingegeben werden. Standardmäßig ist der Anfangstext (*text*) im Editierfeld linksbündig und das Editierfeld besitzt einen Rand. Mehrzeilige Editierfelder besitzen horizontale und vertikale Bildlaufleisten.

Konstruktor

```
TEdit(TWindow* parent, int resourceID, uint textLen, TModule* module = 0);
```

Konstruiert ein *TEdit*-Objekt, das mit einem Editierfeld eines *TDialog*-Objektes assoziiert werden soll. Ruft den *TStatic*-Konstruktor mit identischen Parametern auf. Der *resourceID*-Parameter muß mit einer Edit-Ressource korrespondieren, die Sie definieren können. Aktiviert den Datentransfermechanismus durch den Aufruf von *EnableTransfer*.

Siehe auch: *TStatic::TStatic*

Öffentliche Elementfunktionen

CanUndo

```
bool CanUndo();
```

Gibt TRUE zurück, wenn die letzt Bearbeitung rückgängig gemacht werden kann.

Siehe auch: *TEdit::Undo*

Clear

```
void Clear();
```

Überschreibt die virtuelle Elementfunktion von *TStatic* und löscht den gesamten Text.

ClearModify

```
void ClearModify();
```

Setzt das Änderungs-Flag des Editierfeldes zurück, wodurch *IsModified* FALSE zurückliefert. Das Flag wird gesetzt, wenn der Text geändert wird.

Siehe auch: *TEdit::IsModified*

Copy

```
void Copy();
```

Kopiert den aktuell ausgewählten Text in die Zwischenablage.

Cut

```
void Cut();
```

Löscht den aktuell ausgewählten Text und kopiert ihn in die Zwischenablage.

DeleteLine

```
bool DeleteLine(int lineNumber);
```

Löscht in einem mehrzeiligen Editierfeld den Text in der mit *lineNumber* angegebenen Zeile. Wenn -1 übergeben wird, löscht die Funktion die aktuelle Zeile. *DeleteLine* löscht nicht den Zeilenumbruch und hat keine Einwirkung auf andere Zeilen. Gibt TRUE zurück, wenn der Aufruf erfolgreich ist. Gibt FALSE zurück, wenn *lineNumber* nicht -1 ist und außerhalb des Bereiches liegt oder wenn ein Fehler auftritt.

DeleteSelection

```
bool DeleteSelection();
```

Löscht den aktuell ausgewählten Text und gibt FALSE zurück, wenn kein Text ausgewählt ist.

DeleteSubText

```
bool DeleteSubText(uint startPos, uint endPos);
```

Löscht den Text zwischen den angegebenen Start- und Endpositionen *startPos* und *endPos*. *DeleteSubText* gibt TRUE zurück, wenn der Aufruf erfolgreich ist.

EmptyUndoBuffer

```
void EmptyUndoBuffer();
```

Wenn eine Operation innerhalb des Editierfeldes rückgängig gemacht werden kann, wird das Undo-Flag des Editierfeldes gesetzt. *EmptyUndoBuffer* setzt dieses Flag zurück oder löscht es.

FormatLines

```
inline void FormatLines(bool addEOL);
```

Zeigt an, ob Zeilenendezeichen (Zeilenrücklauf, Zeilenvorschub) bei Textzeilen, die in einem mehrzeiligen Editierfeld umgebrochen werden, hinzugefügt oder entfernt werden sollen. Gibt TRUE zurück, wenn diese Zeichen am Ende umgebrochener Zeilen plaziert werden, und FALSE, wenn sie entfernt werden.

GetFirstVisibleLine

```
int GetFirstVisibleLine() const;
```

Zeigt die oberste sichtbare Zeile in einem Editierfeld an. Bei einzeiligen Editierfeldern ist der Rückgabewert Null. Bei mehrzeiligen Editierfeldern ist der Rückgabewert der Index der obersten sichtbaren Zeile.

GetHandle

```
HLOCAL GetHandle() const;
```

Gibt das Daten-Handle des Puffers zurück, der den Inhalt des Dialogelementfensters enthält.

Diese Funktion ist veraltet und steht unter Presentation Manager nicht zur Verfügung.

Siehe auch: *:TEdit::SetHandle*

GetLine

```
bool GetLine(char far* str, int strSize, int lineNumber);
```

Ermittelt eine Textzeile (dessen Zeilennummer bereitgestellt wird) aus dem Editierfeld und gibt sie in *str* (null-terminiert) zurück. *strSize* zeigt an, wie viele Zeichen zu ermitteln sind. *GetLine* gibt FALSE zurück, wenn der Text nicht ermittelt werden kann, oder wenn der bereitgestellte Puffer zu klein ist.

Siehe auch: *TStatic::GetText*, *TEdit::GetNumLines*, *TEdit::GetLineLength*

GetLineFromPos

```
int GetLineFromPos(uint charPos);
```

Gibt bei einem mehrzeiligen Editierfeld die Zeilennummer zurück, auf der die durch *charPos* angegebene Zeichenposition vorkommt. Wenn *charPos* größer als die Position des letzten Zeichens ist, wird die Nummer der letzten Zeile zurückgegeben. Wenn *charPos* -1 ist, wird die Nummer der Zeile zurückgegeben, die das erste ausgewählte Zeichen enthält. Wenn keine Auswahl existiert, wird die Zeile mit dem Cursor zurückgegeben.

GetLineIndex

```
uint GetLineIndex(int lineNumber);
```

In einem mehrzeiligen Editierfeld gibt *GetLineIndex* die Anzahl von Zeichen zurück, die vor der durch *lineNumber* angegebenen Zeilennummer vorkommen. Wenn *lineNumber* -1 ist, gibt *GetLineIndex* die Nummer der Zeile zurück, die den Cursor enthält.

GetLineLength

```
int GetLineLength(int lineNumber);
```

Bei einem mehrzeiligen Editierfeld gibt *GetLineLength* die Zahl von Zeichen in der durch *lineNumber* angegebenen Zeile zurück. Wenn der Wert -1 ist, arbeitet die Funktion wie folgt: wenn kein Text ausgewählt ist, gibt *GetLineLength* die Länge der Zeile zurück, wo der Cursor positioniert ist. Wenn Text auf der Zeile ausgewählt ist, gibt *GetLineLength* die Zeilenlänge minus der Zahl ausgewählter Zeichen zurück. Wenn sich ausgewählter Text über mehr als eine Zeile streckt, gibt *GetLineLength* die Länge der Zeilen minus der Zahl ausgewählter Zeichen zurück.

GetNumLines

```
int GetNumLines();
```

Gibt die Zahl von Zeilen zurück, die in einem mehrzeiligen Editierfeld eingegeben wurden: 1, wenn das Editierfeld keinen Text enthält (wenn es eine Zeile ohne Text enthält), oder 0, wenn kein Text vorhanden ist oder ein Fehler auftritt.

GetPasswordChar

```
uint GetPasswordChar() const;
```

Gibt das Zeichen zurück, das an Stelle des vom Benutzer eingegebenen Zeichens angezeigt werden soll. Wenn das Editierfeld erzeugt wird und dabei die Eigenschaft ES_PASSWORD angegeben wird, ist das vorgegebene Zeichen für die Anzeige ein Sternchen (*).

Siehe auch: *TEdit::SetPasswordChar*

GetRect

```
void GetRect(TRect& frmtRect) const;
```

Liefert das Formatierrechteck eines mehrzeiligen Editierfeldes.

Siehe auch: *TEdit::SetRect, TEdit::SetRectNP*

GetSelection

```
void GetSelection(uint& startPos, uint& endPos);
```

Gibt die Start- und Endpositionen (*startPos* und *endPos*) des aktuell ausgewählten Textes zurück. Durch Verwendung von *GetSelection* in Verbindung mit *GetSubText* können Sie den aktuell ausgewählten Text ermitteln.

Siehe auch: *TEdit::GetSubText*

GetSubText

```
void GetSubText(char far* str, uint startPos, uint endPos);
```

Ermittelt den Text in einem Editierfeld von den Indizes *startPos* bis *endPos* und gibt ihn in *str* zurück.

Siehe auch: *TEdit::GetSelection*

GetWordBreakProc

```
EDITWORDBREAKPROC GetWordBreakProc() const;
```

Ermittelt die aktuelle Zeilenumbruch-Funktion. Gibt die Adresse der durch die Anwendung definierten Zeilenumbruch-Funktion oder Null zurück, wenn keine existiert .

Siehe auch: *TEdit::SetWordBreakProc*

Insert

```
void Insert(const char far* str);
```

Fügt den in *str* bereitgestellten Text in das Editierfeld am aktuellen Text-Einfügepunkt (Cursor-Position) ein und ersetzt jeden aktuell ausgewählten Text. *Insert* hat Ähnlichkeiten mit *Paste*, sie hat jedoch keine Einwirkung auf die Zwischenablage.

Siehe auch: *TEdit::Paste*

IsModified

```
bool IsModified();
```

Gibt TRUE zurück, wenn der Benutzer den Text im Editierfeld geändert hat.

Siehe auch: *TEdit::ClearModify*

IsValid

```
bool IsValid(bool reportErr = false);
```

Gibt TRUE zurück, wenn der Inhalt des Editierfeldes gültig ist. *reportErr* ist FALSE, so daß *IsValid* standardmäßig kein System- oder Spezial-Meldungsfenster mit einem Fehler-String aus der Standard-String-Tabelle ausgibt.

LockBuffer

```
char far* LockBuffer(uint newsize=0);
```

Sperrt den Puffer des Editierfeldes und gibt einen Zeiger auf den Puffer zurück. Wenn *newsize* mit einem Wert größer als Null übergeben wird, ändert der Puffer seine Größe auf *newsize*. Sie müssen *Unlock* aufrufen, wenn Sie fertig sind.

Siehe auch: *TEdit::UnLockBuffer*

Paste

```
void Paste();
```

Fügt Text aus der Zwischenablage in das Editierfeld am aktuellen Text-Einfügepunkt (Cursor-Position) ein.

Siehe auch: *TEdit::CMEditPaste*

Scroll

```
void Scroll(int horizontalUnit, int verticalUnit);
```

Blättert ein mehrzeiliges Editierfeld horizontal und vertikal unter Verwendung Anzahl Zeichen, die in *horizontalUnit* und *verticalUnit* angegeben sind. Positive Werte resultieren in einem Blättern im Editierfeld nach rechts oder unten, negative Werte in einem Blättern nach links oder oben.

Search

```
int Search(uint startPos, const char far* text,
           bool caseSensitive=false, bool wholeWord=false,
           bool up=false);
```

Führt eine Suche für den bereitgestellten Text mit oder ohne Beachtung der Groß- u. Kleinschreibung durch. Wenn der Text gefunden wird, wird der übereinstimmende Text ausgewählt und *Search* gibt die Startposition des übereinstimmenden Textes zurück. Wenn der Text nicht im Text des Editierfeldes gefunden wird, gibt *Search* -1 zurück. Wenn -1 als *startPos* übergeben wird, dann startet die Suche entweder vom Ende oder vom Anfang des aktuell ausgesuchten Textes. Dies hängt von der Suchrichtung ab.

SetHandle

```
void SetHandle(HLOCAL localMem);
```

Setzt ein Handle auf den Textpuffer, der benutzt wird, um den Inhalt eines mehrzeiligen Editierfeldes aufzunehmen.

Diese Funktion ist veraltet und steht unter Presentation Manager nicht zur Verfügung.

Siehe auch: :*TEdit::GetHandle*

SetPasswordChar

```
void SetPasswordChar(uint ch);
```

SetPasswordChar setzt das Zeichen, das an Stelle des vom Benutzer eingegebenen Zeichens angezeigt wird. Wenn die Eigenschaft ES_PASSWORD angegeben ist, ist das vorgegebene Zeichen für die Anzeige ein Sternchen (*).

Siehe auch: *TEdit::GetPasswordChar*

SetReadOnly

```
void SetReadOnly(bool readOnly);
```

Setzt das Editierfeld für reinen Lesezugriff oder für Lese- und Schreibzugriff um.

SetRect

```
void SetRect(const TRect& frmtRect);
```

Setzt das Formatierrechteck bei einem mehrzeiligen Editierfeld.

Siehe auch: *TEdit::GetRect*, *TEdit::SetRectNP*

SetRectNP

```
void SetRectNP(const TRect& frmtRect);
```

Setzt das Formatierrechteck bei einem mehrzeiligen Editierfeld. Im Gegensatz zu *SetRect* bewirkt *SetRectNP* kein Neuzeichnen des Editierfeldes.

Siehe auch: *TEdit::GetRect*, *TEdit::SetRect*

SetSelection

```
bool SetSelection(uint startPos, uint endPos);
```

Bewirkt die Auswahl eines Textes zwischen den angegebenen Positionen *startPos* und *endPos*, schließt allerdings das Zeichen bei *endPos* nicht mit ein.

SetTabStops

```
void SetTabStops(int numTabs, const int far* tabs);
```

Setzt die Tabulatorpositionen in einem mehrzeiligen Editierfeld.

SetValidator

```
void SetValidator(TValidator* validator);
```

Etabliert das Validierungsobjekt für das Editierfeld.

SetWordBreakProc

```
void SetWordBreakProc(EDITWORDBREAKPROC proc);
```

In einem mehrzeiligen Editierfeld zeigt *SetWordBreakProc* an, daß eine von der Anwendung bereitgestellte Wortumbruch-Funktion die vorgegebene Wortumbruch-Funktion ersetzt hat. Die von der Anwendung bereitgestellte Wortumbruch-Funktion könnte die Wörter im Textpuffer an einem anderen Zeichen umbrechen als am vordefinierten Leerzeichen.

Siehe auch: *TEdit::GetWordBreakProc*

Transfer

```
uint Transfer(void* buffer TTransferDirection direction);
```

Überträgt Informationen für das Editierfeld *TEdit* und schickt Informationen an den *Validator*, wenn ein solcher existiert und die Übertragungsoption gesetzt wurde. *Transfer* kann Typumwandlungen durchführen, wenn Prüfungsobjekte vorhanden sind, beispielsweise wenn *TRangeValidator* Integerwerte überträgt. Der Rückgabewert ist die Größe der Übertragungsdaten (in Bytes).

Undo

```
void Undo();
```

Macht den letzten Bearbeitungsschritt rückgängig.

Siehe auch: *TEdit::CanUndo*, *TEdit::CMEditUndo*

UnlockBuffer

```
void UnlockBuffer(const char far* buffer, bool updateHandle=false);
```

Hebt die Sperre eines gesperrten Editierfeld-Puffers auf. Wenn der Inhalt verändert wurde, sollte *updateHandle* TRUE sein.

Siehe auch: *TEdit::LockBuffer*

ValidationError

```
void ValidatorError();
```

Behandelt Validierungsfehler, die als Ergebnis der Validierung des Editierfeldes auftreten.

Geschütztes Datenelement

Validator

```
TValidator* Validator;
```

Zeigt auf ein Prüfungsobjekt, das in Ihrer abgeleiteten Klasse konstruiert wurde, um Eingabetext zu überprüfen. Wenn kein Prüfungsobjekt existiert, ist *Validator* Null.

Geschützte Elementfunktionen

CanClose

```
bool CanClose();
```

Überprüft, ob alle untergeordneten Fenster geschlossen werden können, bevor das aktuelle Fenster geschlossen wird. Wenn irgendein untergeordnetes Fenster FALSE zurückgibt, gibt auch *CanClose* FALSE zurück und beendet den Prozeß. Wenn alle untergeordneten Fenster geschlossen werden können, gibt *CanClose* TRUE zurück.

CmCharsEnable

```
void CmCharsEnable(TCommandEnabler& commandHandler);
```

Bestimmt, ob der Menüeintrag *Clear* für den aktuell ausgewählten Text aktiviert ist.

CmEditClear

```
void CmEditClear();
```

Antwortet automatisch auf eine Menüauswahl mit einer Menü-ID CM_EDITCLEAR, indem *Clear* aufgerufen wird.

Siehe auch: *TStatic::Clear*

CmEditCopy

```
void CmEditCopy();
```

Antwortet automatisch auf eine Menüauswahl mit einer Menü-ID CM_EDITCOPY, indem *Copy* aufgerufen wird.

Siehe auch: *TEdit::Copy*

CmEditCut

```
void CmEditCut();
```

Antwortet automatisch auf eine Menüauswahl mit einer Menü-ID CM_EDITCUT, indem *Cut* aufgerufen wird.

Siehe auch: *TEdit::Cut*

CmEditDelete

```
void CmEditDelete();
```

Antwortet automatisch auf eine Menüauswahl mit einer Menü-ID CM_EDITDELETE, indem *DeleteSelection* aufgerufen wird.

Siehe auch: *TEdit::DeleteSelection*

CmEditPaste

```
void CmEditPaste();
```

Antwortet automatisch auf eine Menüauswahl mit einer Menü-ID CM_EDITPASTE, indem *Paste* aufgerufen wird.

Siehe auch: *TEdit::Paste*

CmEditUndo

```
void CmEditUndo();
```

Antwortet automatisch auf eine Menüauswahl mit einer Menü-ID CM_EDITUNDO, indem *Undo* aufgerufen wird.

Siehe auch: *TEdit::Undo*

CmModEnable

```
void CmModEnable(TCommandEnabler& commandHandler);
```

Bestimmt, ob der Menüeintrag *Undo* für den aktuell ausgewählten Text aktiviert ist.

CmPasteEnable

```
void CmPasteEnable(TCommandEnabler& commandHandler);
```

Bestimmt, ob der Menüeintrag *Paste* für den aktuell ausgewählten Text aktiviert ist.

CmSelectEnable

```
void CmSelectEnable(TCommandEnabler& commandHandler);
```

Bestimmt, ob die Menüeinträge *Cut, Copy* oder *Delete* für den aktuell ausgewählten Text aktiviert sind.

ENErrSpace

```
void ENErrSpace();
```

Läßt einen Piepton in Antwort auf eine Fehler-Benachrichtigungsbotschaft erklingen, die gesendet wird, wenn das Editierfeld erfolglos versucht, mehr Speicherplatz zu reservieren.

EvChar

```
void EvChar(uint key, uint repeatCount, uint flags);
```

Überprüft den in das Editierfeld eingegebenen Text. Wenn die Eingabe inkorrekt ist, wird der originale Text wieder zurückgespielt. Andernfalls wird der überprüfte und modifizierte Text zurück in das Editierfeld plaziert, wodurch die Ergebnisse des automatischen Füllens (sofern welche vorliegen) dargestellt werden können. Wenn *IsValidInput* aufgerufen wird, hat der Parameter *SupressFill* standardmäßig den Wert FALSE, so daß der String verändert werden kann.

EvGetDlgCode

```
uint EvGetDlgCode(MSG far* msg);
```

Antwortet auf die WM_GETDLGCODE-Botschaften, die an ein Dialogfenster, das mit einem Dialogelement assoziiert ist, geschickt werden. *EvGetDlgCode* erlaubt es dem Dialogverwalter, eine Botschaft abzufangen, die normalerweise an ein Dialogelement geht. Er fragt dann das Dialogelement ab, ob es diese Botschaft verarbeiten will. Ist dies nicht der Fall, verarbeitet der Dialogverwalter die Botschaft. Der Parameter *msg* gibt die Art der Botschaft an, die an den Dialogfenster-Verwalter geschickt wird - beispielsweise eine Dialogelement-, Befehls- oder Editier-Botschaft.

Wenn das Editierfeld eine gültige Eingabe enthält, dann sind Tabulatoren für einen Wechsel des Fokus erlaubt und ein DLGC_WANTTABS-Code wird zurückgegeben.

Siehe auch *DLGC_xxxx* Dialogelement-Botschafts-Konstanten

EvKeyDown

```
void EvKeyDown(uint key, uint repeatCount, uint flags);
```

EvKeyDown übersetzt den virtuellen Tasten-Code in eine Bewegung. *key* zeigt den virtuellen Tasten-Code der gedrückten Taste an. *repeatCount* enthält die Zahl, wie oft dieselbe Taste gedrückt wurde. *flags* enthält eine der Botschaften, die für die Modus-Anzeigen in den Code einer virtuellen Taste (VK = virtual Key) übersetzt wird. Wenn die Tabulatortaste an das Editierfeld gesendet wird, prüft *EvKeyDown* die Gültigkeit, bevor der Fokus geändert werden kann.

EvKillFocus

```
void EvKillFocus(HWND hWndGetFocus);
```

EvKillFocus antwortet auf eine WM_KILLFOCUS-Botschaft, die an ein Fenster, das die Tastatureingabe verliert, gesendet wurde. Es versteckt und löscht dann den Cursor. *EvKillFocus* überprüft den Text immer dann, wenn der Fokus abgegeben werden soll, und behält den Fokus bei, wenn der Text nicht gültig ist. Es löscht den Fokus nicht, wenn eine andere Anwendung, ein »Abbruch«-Schalter oder ein »OK«-Schalter (in diesem Fall wird *CanClose* aufgerufen, um den Text zu überprüfen), den Fokus besitzt.

GetClassName

```
virtual char far* GetClassName();
```

Gibt den Namen der Windows-Registrierungsklasse von *TEdit*, »EDIT«, zurück.

Siehe auch: *TWindow::GetClassName*

SetupWindow

```
virtual void SetupWindow();
```

Wenn das Datenelement *textLen* nicht Null ist, begrenzt *SetupWindow* die Zahl von Zeichen, die in das Editierfeld eingegeben werde können, auf *textLen* -1.

Siehe auch: *TStatic::TextLen*, *TWindow::SetupWindow*

Anworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_COMMAND (CM_EDITCLEAR, CmEditClear)	CmEditClear
EV_COMMAND (CM_EDITCOPY, CmEditCopy)	CmEditCopy
EV_COMMAND (CM_EDITCUT, CmEditCut)	CmEditCut
EV_COMMAND (CM_EDITDELETE, CmEditDelete)	CmEditClear
EV_COMMAND (CM_EDITPASTE, CmEditPaste)	CmEditPaste
EV_COMMAND (CM_EDITUNDO, CmEditUndo)	CmEditUndo
EV_COMMAND_ENABLE(CM_EDITCLEAR, CmCharsEnable)	CmCharsEnable
EV_COMMAND_ENABLE(CM_EDITCOPY, CmSelectEnable)	CmSelectEnable
EV_COMMAND_ENABLE(CM_EDITCUT, CmSelectEnable)	CmSelectEnable

EV_COMMAND_ENABLE(CM_EDITDELETE, CmSelectEnable)	CmSelectEnable
EV_COMMAND_ENABLE(CM_EDITPASTE, CmPasteEnable)	CmPasteEnable
EV_COMMAND_ENABLE(CM_EDITUNDO, CmModEnable)	CmModEnable
EV_NOTIFY_AT_CHILD (EN_ERRSPACE, ENErrSpace)	ENErrSpace
EV_WM_CHAR	EvChar
EV_WM_GETDLGCODE	EvGetdlgcode
EV_WM_KEYDOWN	EvKeydown
EV_WM_KILLFOCUS	EvKillFocus
EV_WM_CHILDINVALID	EvChildInvalid

Klasse TEditFile editfile.h

TEditFile ist ein Fenster zum Bearbeiten von Dateien. Datenelemente und Elementfunktionen von *TEditFile* verwalten das Datei-Dialogfenster und automatische Antworten auf Dateikommandos wie das Öffnen, Lesen, Schreiben, Speichern und Speichern-unter. *TEditFile* ist streamfähig.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TEditFile(TWindow* ()t = 0, int Id = 0, const char far* text = 0,
          const char far* fileName = 0, TModule* module = 0);
```

Konstruiert ein *TEditFile*-Fenster mit gegebene übergeordnetem Fenster, Ressourcen-ID (*Id*), Text, Dateinamen und Modul-ID. Setzt *Filename* auf *fileName*.

Destruktor

```
~TEditFile();
```

Gibt reservierten Speicher frei, der für den Namen des *TEditFile* benötigt wurde.

Öffentliche Datenelemente

FileData

```
TOpenSaveDialog::TData FileData;
```

Enthält Information über die Auswahl der Benutzers zum Öffnen und Speichern von Dateien.

Siehe auch: *TOpenSaveDialog*::Struktur *TData*

FileName

```
char far* FileName;
```

Enthält den Namen der bearbeiteten Datei.

Öffentliche Elementfunktionen

CanClear

```
virtual bool CanClear();
```

Gibt TRUE zurück, wenn der Text des assoziierten Editierfeldes gelöscht werden kann.

CanClose

```
virtual bool CanClose();
```

Gibt TRUE zurück, wenn das Editierfenster geschlossen werden kann.

CmFileNew

```
void CmFileNew();
```

Ruft *NewFile* als Antwort auf ein eintreffendes Datei-Neu-Kommando mit einem Kommandobezeichner CM_FILENEW auf.

Siehe auch: *TEditFile::NewFile*

CmFileOpen

```
void CmFileOpen();
```

Ruft *Open* als Antwort auf ein eintreffendes Datei-Öffnen-Kommando mit einem Kommandobezeichner CM_FILEOPEN auf.

Siehe auch: *TEditFile::Open*

CmFileSave

```
void CmFileSave();
```

Ruft *Save* als Antwort auf ein eintreffendes Datei-Speichern-Kommando mit einem Kommandobezeichner CM_FILESAVE auf.

Siehe auch: *TEditFile::Save*

CmFileSaveAs

```
void CmFileSaveAs();
```

Ruft *SaveAs* als Antwort auf ein eintreffendes Datei-Speichern-unter-Kommando mit einem Kommandobezeichner CM_FILESAVEAS auf.

Siehe auch: *TEditFile::SaveAs*

NewFile

```
void NewFile();
```

Startet die Bearbeitung einer neuen Datei nach dem Aufruf von *CanClear*, um festzustellen, ob ein Löschen des Textes im Editor erlaubt ist.

Siehe auch: *TEditFile::CanClear*

Open

```
Open();
```

Öffnet eine neue Datei, nachdem überprüft wurde, ob ein Löschen des Textes im *Editor* erlaubt ist. Ruft *CanClear* auf und zeigt, wenn diese Funktion TRUE zurückgibt, ein Datei-Dialogfenster an, um den Namen der neuen Datei vom Benutzer zu erfragen. Ruft *ReplaceWith* auf, um den Namen der neuen Datei zu übergeben.

Siehe auch: *TEditFile::CanClear*, *TEditFile::ReplaceWith*

Read

```
bool Read(const char far* fileName=0);
```

Liest den Inhalt der vorher angegebenen Datei in den *Editor*. Gibt TRUE zurück, wenn die Leseoperation erfolgreich ist.

ReplaceWith

```
void ReplaceWith(const char far* fileName);
```

Ruft *SetFileName* und *Read* auf, um die aktuell bearbeitete Datei durch die Datei zu ersetzen, deren Name bereitgestellt wird.

Siehe auch: *TEditFile::SetFileName*, *TEditFile::Read*

Save

```
bool Save();
```

Speichert die Änderungen am Inhalt vom *Editor* in der Datei. Wenn *Editor->IsModified* FALSE zurückgibt, gibt *Save* TRUE zurück. Damit wird angezeigt, daß seit dem letzten Öffnen oder Speichern keine Änderungen mehr stattgefunden haben.

Siehe auch: *TEditFile::SaveAs*, *TEditFile::Write*

SaveAs

```
bool SaveAs();
```

Speichert den Inhalt des *Editor* in eine Datei, dessen Name vom Benutzer erfragt wird. Dazu wird ein Datei-Speichern-Dialogfenster ausgeführt. Wenn der Benutzer »OK« auswählt, ruft *SaveAs* die Funktionen *SetFileName* und *Write* auf. Gibt TRUE zurück, wenn die Datei gespeichert wurde.

Siehe auch: *TEditFile::SetFileName*, *TEditFile::Write*

SetFileName

```
void SetFileName (const char far* fileName);
```

Setzt *FileName* und aktualisiert den Fenstertitel.

Write

```
bool Write(const char far* fileName=0);
```

Speichert den Inhalt des *Editor* in eine Datei, dessen Name durch *FileName* angegeben ist. Gibt TRUE zurück, wenn die Schreiboperation erfolgreich ist.

Geschützte Elementfunktionen

SetupWindow

```
void SetupWindow();
```

Erzeugt das *Editor*-Editierfeld der Editfensters durch den Aufruf von *TEditFile::SetupWindow*. Setzt den Fenstertitel auf *FileName*, sofern verfügbar. Andernfalls setzt sie den Namen auf »Untitled«.

Siehe auch: *TEditFile::SetFileName*, *TEditFile::Read*

Anworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_COMMAND (CM_FILENEW, CmFileNew)	CmFileNew
EV_COMMAND (CM_FILEOPEN, CmFileOpen)	CmFileOpen
EV_COMMAND (CM_FILESAVE, CmFileSave)	CmFileSave
EV_COMMAND (CM_FILESAVEAS, CmFileSaveAs)	CmFileSaveAs

Klasse TEditSearch editsear.h

TEditSearch ist ein Editierfeld, das auf die Menükommandos *Suchen*, *Ersetzen* und *Weitersuchen* antwortet. Diese Klasse ist streamfähig.

Öffentliche Konstruktoren

Konstruktor

```
TEditSearch(TWindow* parent = 0, int Id = 0, const char far* text = 0,
int x = 0, int y = 0, int w = 0, int h = 0, TModule* module = 0);
```

Konstruiert ein *TEditSearch*-Objekt bei gegebenem übergeordnetem Fenster, Ressourcen-ID und Zeichen-String (*text*).

Öffentliche Datenelemente

SearchCmd

```
uint SearchCmd;
```

Enthält den Suchen-Kommandobezeichner, der ein Dialogfenster geöffnet hat, wenn eines offen ist.

SearchData

```
TFindReplaceDialog::TData SearchData;
```

Die Struktur *SearchData* definiert den Text-String zum Suchen, den zum Ersetzen und die Größe der Textpuffers.

Siehe auch: *TFindReplaceDialog::TData*

SearchDialog

```
TFindReplaceDialog* SearchDialog;
```

Enthält Dialogfenster-Informationen zum Suchen und Ersetzen (wie den zu suchenden und zu ersetzenden Text) sowie Einstellungen von Markierungsfeldern.

Öffentliche Elementfunktionen

CmEditFind

```
void CmEditFind();
```

Öffnet ein *TFindDialog* als Antwort auf ein eintreffendes Suchen-Kommando mit einem CM_EDITFIND-Kommando.

CmEditFindNext

```
void CmEditFindNext();
```

Antwortet auf ein eintreffendes Weitersuchen-Kommando mit einem CM_EDITFINDNEXT-Kommandobezeichner, indem *DoSearch* aufgerufen wird, um die Suchoperation zu wiederholen.

Siehe auch: *TEditSearch::DoSearch*

CmEditReplace

```
void CmEditReplace();
```

Öffnet ein *TReplaceDialog* als Antwort auf ein eintreffendes Ersetzen-Kommando mit einem CM_EDITREPLACE-Kommando.

DoSearch

```
void DoSearch();
```

Führt eine Operation zum Suchen und Ersetzen durch, die auf der Information in *SearchData* basiert.

Siehe auch: *TFindReplaceDialog::TData*

EvFindMsg

```
LRESULT EvFindMsg(WPARAM, LPARAM);
```

Antwortet auf eine Botschaft, die von einem nicht-modalen Dialogfenster zum Suchen und Ersetzen gesendet wurde. Ruft *DoSearch* auf, um die Suche fortzusetzen, wenn der Text nicht gefunden wurde oder das Ende des Dokumentes nicht erreicht wurde.

Siehe auch: *TEditSearch::DoSearch*

SetupWindow

```
void SetupWindow();
```

Hinterlegt eine CM_EDITFIND- oder CM_EDITREPLACE-Botschaft, um ein nicht-modales Dialogfenster zum Suchen und Ersetzen erneut zu öffnen. Ruft *TEdit::SetupWindow* auf.

Siehe auch: *Tedit::SetupWindow*

Anworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_COMMAND(CM_EDITFIND, CmEditFind)	CmEditFind
EV_COMMAND(CM_EDITFINDNEXT, CmEditFindNext)	CmEditFindNext
EV_COMMAND(CM_EDITREPLACE, CmEditReplace)	CmEditReplace
EV_REGISTERED(FINDMSGSTRING, EvFindMsg)	EvFindMsg

Klasse TEditView — editview.h

TEditView ist von *TView* und *TEditSearch* abgeleitet und bietet ein View-Rahmenobjekt für die ObjectWindows Texteditklasse (*TEdit*). Eine streamfähige Klasse *TEditView* umfaßt mehrere Ereignisbehandlungsfunktionen, die Botschaften zwischen einem Dokument und seinen Views behandeln.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TEditView(TDocument& doc, TWindow* parent = 0);
```

Erzeugt ein *TEditView*-Objekt, das mit dem angegebenen Dokument und dem übergeordneten Fenster assoziiert ist. Setzt *Attr.AccelTable* auf

IDA_EDITVIEW, um die Edit-View zu identifizieren. Setzt *TView::View-Menu* auf das neue *TMenuDescr*-Objekt für diese View.

Destruktor

```
~TEditView();
```

Entfernt ein *TEditView*-Objekt.

Öffentliche Elementfunktionen

CanClose

```
bool CanClose();
```

Gibt einen Wert ungleich Null zurück, wenn die View geschlossen werden kann.

Siehe auch: *TEditSearch::CanClose*

Create

```
bool Create();
```

Überschreibt *TWindow::Create* und ruft *TEditSearch::Create* auf, um das Fenster der View zu erzeugen. Ruft *GetDocPath* auf, um zu bestimmen, ob die Datei neu ist oder ob sie bereits Daten enthält. Wenn Daten existieren, ruft sie *LoadData* auf, um die Daten der View hinzuzufügen. Wenn das Fenster der View nicht erzeugt werden kann, zeigt *Create* an, daß die View ungültig ist.

GetViewName

```
LPCSTR GetViewName();
```

Überschreibt *TView::GetViewName* und gibt den beschreibenden Namen der Klasse (*StaticName*) zurück.

Siehe auch: *TEditView::StaticName*, *TView::GetViewName*

GetWindow

```
TWindow* GetWindow();
```

GetWindow überschreibt *GetWindow* in *TView* und gibt **this** als ein *TWindow* zurück.

Siehe auch: *TView::GetWindow*

PerformCreate

```
void Perform Create(int menuOrId);
```

Reserviert nach Bedarf Speicherplatz, so daß *TEditView* Dateien bis zu einschließlich 30.000 Bytes behandeln kann.

SetDocTitle

```
bool SetDocTitle(LPCSTR docname, int index);
```

Überschreibt *TView::SetDocTitle* und leitet den Titel an ihre Basisklasse *TEditSearch* weiter. *index* ist die Nummer der View, die in der Titelleiste angezeigt wird. *docname* ist der Name des Dokuments, der im View-Fenster angezeigt wird.

Siehe auch: *TWindow::SetDocTitle*, *TView::SetDocTitle*

StaticName

```
static LPCSTR StaticName();
```

Gibt »Edit View« als den beschreibenden Namen der Klasse für das *View/Select*-Menü zurück.

Geschützte Datenelemente

Origin

```
long Origin;
```

Enthält die Dateiposition am Startpunkt der Anzeige.

Geschützte Elementfunktionen

EvNCDestroy

```
void EvNCDestroy();
```

EvNcDestroy wird intern von *TEditView* für die Speicherverwaltung benutzt.

Dieses Element steht unter Presentation Manager nicht zur Verfügung.

LoadData

```
bool LoadData();
```

LoadData liest die View aus einem Stream und schließt die Datei. Sie gibt einen Wert ungleich Null zurück, wenn die View erfolgreich geladen wurde. Wenn die Datei nicht gelesen werden kann, hinterlegt sie einen Fehler und gibt Null zurück.

VnCommit

```
bool VnCommit(bool force);
```

VnCommit übergibt die in der View vorgenommenen Änderungen an das Dokument. Wenn *force* einen Wert ungleich Null hat, werden alle Daten, auch wenn sie unverändert sind, im Dokument gespeichert.

Siehe auch: *TEditView::vnRevert*, *vnxxxx View-Benachrichtigungs*-Konstanten

VnDocClosed

```
bool VnDocClosed(int omode);
```

VnDocClosed zeigt an, daß das Dokument geschlossen wurde. *mode* ist eine der *ofxxxx Dokument-offen*-Konstanten.

Siehe auch: *ofxxxx Dokument-offen*-Aufzählungstyp, *vnxxxx View-Benachrichtigungs*-Konstanten

VnIsDirty

```
bool VnIsDirty();
```

VnIsDirty gibt einen Wert ungleich Null zurück, wenn Änderungen, die an den Daten in der View vorgenommen wurden, nicht im Dokument gespeichert wurden. Andernfalls gibt die Funktion Null zurück.

Siehe auch: *vnxxxx View-Benachrichtigungs*-Konstanten

VnIsWindow

```
bool VnIsWindow(HWND hWnd);
```

VnIsWindow gibt einen Wert ungleich Null zurück, wenn das in *hWnd* übergebene Handle des Fensters dasselbe ist wie das des Anzeigefensters für die View.

VnRevert

```
bool VnRevert(bool clear);
```

VnRevert hat einen Wert ungleich Null, wenn die an der View vorgenommenen Änderungen freigegeben und die Daten aus dem Dokument wieder in die View eingespielt werden sollen. Wenn *clear* ungleich Null ist, werden die Daten gelöscht und nicht in die View wiedereingespielt.

Siehe auch: *TEditView::VnCommit*

Anworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_VN_COMMIT	VnCommit
EV_VN_DOCCLOSED	VnDocClosed
EV_VN_ISDIRTY	VnIsDirty
EV_VN_ISWINDOW	VnIsWindow
EV_WM_NCDESTROY	EvNcDestroy
EV_VN_REVERT	VnRevert

Typdefinition TEqualOperator — eventhan.h

```
typedef bool(*TEqualOperator)(TGenericTableEntry __RTFAR&, TEventInfo&);
```

TequalOperator dient der Durchführung spezieller Formen von Suchvorgängen und der Erleichterung beim Auffinden von Antworttabelleneinträgen. *TEqualOperator* vergleicht ein spezielles Botschaftsereignis (*TEventInfo&*) mit einem Antworttabelleneintrag (*TGenericTableEntry*), um zu bestimmen, ob sie übereinstimmen.

Siehe auch: *TResponseTableEntry*

Klasse TEventHandler::TEventInfo — eventhan.h

Eine eingebettete Klasse mit Namen *TEventInfo* liefert spezifische Informationen über den gesendeten Botschaftstyp, die Klasse, welche die zu behandelnde Funktion enthält, den entsprechenden Antworttabelleneintrag und die Zuteilungsfunktion, die die Botschaft verarbeitet.

Öffentliche Konstruktoren

Konstruktor

```
TEventInfo(uint msg, uint id = 0) : Msg(msg), Id(id);
```

Konstruiert ein *TEventInfo*-Objekt mit der angegebenen ID und Botschaftstyp.

Aufzählungstyp TEventStatus

window.h

```
enum (esPartial, esComplete);
```

Ereignisstatuskonstanten zeigen den Status einer vermischten Implementierung von Fensterereignissen an. So geben zum Beispiel *DoKeyDown* und *DoSetFocus,* abhängig von der Implementierung eines Tastaturereignisses, eine der *TEventStatus*-Konstanten zurück.

Ereignisstatuskonstanten

Konstante	Bedeutung
esPartial	Zusätzliche Behandlungsroutinen können aufgerufen werden.
esComplete	Es werden keine zusätzlichen Behandlungsroutinen benötigt.

Öffentliche Datenelemente

Entry

```
TGenericTableEntry __RTFAR *Entry;
```

Zeigt auf den Antworttabelleneintrag, zum Beispiel *EvActivate*.

Id

```
const uint Id;
```

Enthält die Ressourcen-ID des Menüs oder der Tastenkürzel (CM_xxxx) für die Elementfunktion zur Botschaftsbeantwortung.

Msg

```
const uint Msg;
```

Enthält den gesendeten Botschaftstyp. Diese Botschaften können Kommandobotschaften, Botschaften, die auf der ID untergeordneter Fenster basieren, auf Benachrichtigungen basierende Botschaften wie zum Beispiel LBN_SELCHANGE oder Windows-Botschaften wie zum Beispiel LBUTTONDOWN sein.

Object

```
GENERIC *Object;
```

Zeigt auf das Objekt, das die zu behandelnde Funktion enthält.

Klasse TEventHandler eventhan.h

TEventHandler ist eine Basisklasse, von der aus Sie Klassen für die Behandlung von Botschaften ableiten können. *TEventHandler* führt insbesondere die folgenden ereignisbezogenen Aufgaben aus:

1. Analysiert eine Fensterbotschaft.
2. Durchsucht die Antworttabelleneinträge der Klasse nach einer geeigneten Ereignisbehandlungsroutine.
3. Gibt die Botschaft an die ausgewählte Ereignisbehandlungsroutine weiter.

Die meisten ObjectWindows Klassen sind von *TEventHandler* abgeleitet und erben daher dieses Verhalten bezüglich der Ereignisbehandlung. Außerdem kann jede benutzerdefinierte, von *TEventHandler* abgeleitete Klasse Botschaftsantwortfunktionen behandeln, die mit einer Fensterbotschaft assoziiert sind.

Siehe auch Makro *DECLARE_RESPONSE_TABL*

Öffentliche Elementfunktionen

Dispatch

```
virtual LRESULT Dispatch(TEventInfo&, WPARAM, LPARAM = 0);
```

Nimmt die Botschaftsdaten von *Msg* und teilt sie der korrekten Ereignisbehandlung zu.

Find

```
virtual bool Find(TEventInfo&, TEqualOperator = 0);
```

Durchsucht die Liste der Anworttabelleneinträge nach einer Übereinstimmung. Da *TEventHandler* keine Einträge besitzt, gibt die Implementierung von *TEventHandler* dieser Routine FALSE zurück.

RegisterWindowMessage

```
static uint RegisterWindowMessage(const char far* lpszString);
```

(Nur unter Presentation Manager) Registriert eine neue Fensterbotschaft, die durch *lpszString* benannt wird. Gibt die Nummer zurück, die die neue Botschaft repräsentiert. Verwenden Sie diese Funktion, wenn alle Anwendungen die Botschaft kennen müssen. Verwenden Sie WM_USER plus einen Offset, wenn die Botschaft privat ist.

Geschützte Elementfunktionen

SearchEntries

```
bool SearchEntries(TGenericTableEntry __RTFAR* entries, TEventInfo&,
                   TEqualOperator);
```

Durchsucht die Einträge in der Antworttabelle nach einem Eintrag, der mit *TEventInfo* übereinstimmt, oder einen Eintrag, der von *TEqualOperator* als Übereinstimmung erkannt wird.

Klasse TFileDocument filedoc.h

Als eine von *TDocument* abgeleitete Klasse öffnet und schließt *TFileDocument* Views und stellt ferner Stream-Unterstützung für Views zur Verfügung. Streams werden oberhalb von DOS-Dateien mit Hilfe der Dateidienstprogramme von Windows erzeugt. *TFileDocument* besitzt Elementfunktionen, die die Verarbeitung von Datei-Öffnen- und Datei-Neu-Botschaften nach dem Konstruieren einer View fortsetzen. Sie können eine Unterstützung für spezialisierte Dateitypen hinzufügen, indem Sie Klassen von *TFileDocument* ableiten. *TFileDocument* erleichtert diesen Prozeß dadurch, daß sie den aktuellen Prozeß des Speichers von Dateitypen versteckt.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TFileDocument(TDocument* parent = 0);
```

Konstruiert ein *TFileDocument*-Objekt mit dem optionalen übergeordneten Dokument.

Destruktor

```
~TFileDocument(TDocument* parent = 0)
```

Entfernt ein *TFileDocument*-Objekt.

Typdefinition

TFileDocProp

```
enum TFileDocProp {PrevProperty, CreateTime, ModifyTime, AccessTime,
StorageSize, FileHandle, NextProperty};
```

Enthält Konstanten, die die folgenden Eigenschaften des Dokuments definieren:

Wert	Bedeutung
PrevProperty	*TDocument::NextProperty-1*. Dies ist der erste Wert für View- und Dokument-Objekte.
CreateTime	Zeitpunkt, an dem die View oder das Dokument erzeugt wurden.
ModifyTime	Zeitpunkt, an dem die View oder das Dokument geändert wurden.
AccessTime	Letzter Zugriffszeitpunkt auf die View oder das Dokument.
StorageSize	Unsigned long, der die Speichergröße enthält.
FileHandle	Das Datei-Handle der Plattform.
NextProperty	Der letzte Wert für die Eigenschaft.

Klassen, die von *TDocument* und *TView* abgeleitet sind, können diese generischen Dateieigenschaftswerte verwenden.

Öffentliche Elementfunktionen

Close

```
bool Close();
```

Schließt das Dokument, löscht und entfernt allerdings nicht irgendwelche assoziierten Views. Vor dem Schließen des Dokumentes ruft *Close* die Funktion *Close* von *TDocument* auf, um sicher zu stellen, das alle untergeordneten Dokumente geschlossen werden. Wenn irgendein untergeordnetes Dokument offen ist, gibt *Close* Null zurück und schließt das Dokument nicht. Wenn alle untergeordneten Dokumente geschlossen sind, überprüft die Funktion, ob irgendwelche assoziierten Streams offen sind. Trifft dies zu, gibt die Funktion Null zurück und schließt das Dokument nicht. Wenn keine offenen Streams existieren, schließt die Funktion die Datei.

Siehe auch: *TDocument::Close*

Commit

```
bool Commit(bool clear = false);
```

Ruft *TDocument_Commit* auf und löscht das Datenelement *DirtyFlag* von *TDocument*. Damit wird angezeigt, daß keine ungesicherten Änderungen mehr am Dokument existieren.

Siehe auch: *TDocument::Commit*

FindProperty

```
int FindProperty(const char far* name);
```

FindProperty ermittelt den Eigenschaftsindex bei gegebenem Eigenschaftsnamen (*name*). Gibt Null zurück, wenn der Name nicht gefunden wurde.

Siehe auch: *pfxxxx Eigenschaftsattribut*-Konstanten

GetProperty

```
int GetProperty(int index, void far* dest, int textlen = 0);
```

Überschreibt *TDocument::GetProperty* und ermittelt die Eigenschafts-ID für das aktuelle Dateidokument.

Siehe auch: *pfxxxx Eigenschaftsattribut*-Konstanten

InStream

```
TInStream* InStream(int mode, const char far* strmId = 0);
```

Überschreibt *TDocument_InStream* und bietet generische Eingabe für das spezielle Speichermedium. *InStream* gibt einen Zeiger auf ein *TInStream* zurück. *mode* ist eine Kombination der in iostream.h definierten ios-Bits. *strmId* wird für Dateidokumente nicht benutzt. Die View liest Daten aus dem Dokument als Stream oder über Stream-Funktionen.

Siehe auch: *TFileDocument::OutStream*

IsOpen

```
bool IsOpen();
```

Ist ungleich Null, wenn das Dokument oder irgendwelche Streams offen sind.

Open

```
bool Open(HFILE fhdl);
```

Öffnet ein Dateidokument unter Verwendung eines existierenden Datei-Handle. Setzt *TDocument_OpenMode* auf PREV_OPEN und lesen/schreiben. Setzt den Dokumentpfad auf Null. Setzt *FHd* auf *fhdl*. Gibt immer einen Wert ungleich Null zurück.

Open

```
bool Open(int mode, const char far* path = 0);
```

Überschreibt *TDocument_Open* und öffnet die Datei unter Verwendung des angegebenen Pfads *path*. Wenn die Datei bereits geöffnet ist, gibt die Funktion Null zurück. Ruft *TDocument_SetDocPath* auf, um den Verzeichnispfad zu setzen. Wenn *omode* nicht Null ist, setzt die Funktion *TDocument_OpenMode* auf *omode*. Wenn die Datei nicht geöffnet werden kann, gibt die Funktion Null zurück.

Siehe auch: *TDocument::SetDocPath*, *TDocument::Open*, *ofxxxx Dokument-offen* Aufzählungstyp

OutStream

```
TOutStream* OutStream (int mode, const char far* strmId = 0);
```

Überschreibt *TDocument_OutStream* und bietet generische Eingabe für das spezielle Speichermedium. *OutStream* gibt einen Zeiger auf ein *TOutStream* zurück. *mode* ist eine Kombination der in iostream.h definierten ios-Bits. *strmId* wird für Dateidokumente nicht benutzt. Stattdessen liest die View Daten aus dem Dokument über Stream-Funktionen.

Siehe auch: *TFileDocument::InStream*

PropertyFlags

```
int PropertyFlags(int index);
```

Gibt die Eigenschaftsattributkonstanten (*pfGetText*, *pfHidden* usw.) zurück.

Siehe auch: *pfxxxx Eigenschaftsattribut*-Konstanten

PropertyName

```
const char* PropertyName(int index);
```

Gibt den Textnamen der Eigenschaft bei gegebenem Indexwert zurück.

Siehe auch: *pfxxxx Eigenschaftsattribut*-Konstanten

Revert

```
bool Revert(bool clear = false);
```

Ruft *TDocument_Revert* auf, um die Views zu benachrichtigen, damit sie ihre Daten aktualisieren. Wenn *clear* FALSE ist, werden die Daten wiedereingespielt anstatt sie zu löschen.

Siehe auch: *TFileDoc::Commit*

SetProperty

```
bool SetProperty(int index, const void far* src);
```

Setzt die Eigenschaftsdaten, die in einem nativen Datentyp vorliegen müssen (entweder String oder binär).

Siehe auch: *pfxxxx Eigenschaftsattribut*-Konstanten

Geschützte Datenelemente

FHdl

```
HFILE FHdl;
```

Enthält das Datei-Handle auf ein offenes Dateidokument.

Geschützte Elementfunktionen

CloseThisFile

```
void CloseThisFile(HFILE fhdl, into omode);
```

Wenn die assoziierte Datei durch *TFileDocument* geöffnet wurde, schließt *CloseThisFile* das Datei-Handle. Ruft *TDocument_NotifyView* auf, um alle Views zu benachrichtigen, daß das Dateidokument geschlossen wurde.

Siehe auch: *ofxxxx Dokument-offen*-Aufzählungstyp

OpenThisFile

```
HFILE OpenThisFile(int omode, const char far* name, streampos* pseekpos);
```

Öffnet das Dateidokument nach der Überprüfung des Dateizugriffsmodus (*omode*). Wenn ein Dateimodus nicht als Lesen, Schreiben oder Lesen und Schreiben angegeben ist, gibt die Funktion Null zurück.

Siehe auch: *ofxxxx Dokument-offen*-Aufzählungstyp, *shxxxx-Dokumentzugriffsmodi*

Klasse TFileOpenDialog — opensave.h

TFileOpenDialog ist ein modales Dialogfenster, das Ihnen die Angabe des Namens einer zu öffnenden Datei ermöglicht. Verwenden Sie dieses Dialogfenster, um auf einen CM_FILEOPEN-Befehl zu antworten, der erzeugt wird, wenn ein Anwender Datei/Öffnen aus einem Menü wählt. *TFileOpenDialog* verwendet die Struktur *TOpenSave::TData*, um das Datei-Öffnen-Dialogfenster zu initialisieren.

Öffentliche Konstruktoren

Konstruktor

```
TFileOpenDialog(TWindow* parent, TData& data, TResID templateID = 0,
                const char far* title = 0, TModule* module = 0);
```

Konstruiert und initialisiert das *TFileOpen*-Objekt, das auf der Information in der Datenstruktur *TOpenSaveDialog::TData* basiert. Das Argument *parent* zeigt auf das übergeordnete Fenster des Dialogfensters. *data* stellt einen Bezug auf das Objekt *TData* dar. *templateID* ist die Kennung für ein

anwenderdefiniertes Template. *title* ist ein optionaler Titel. *module* zeigt auf die Modul-Instanz.

Siehe auch: *TOpenSaveDialog::TData, TOpenSaveDialog, TResID, TModule*

Öffentliche Elementfunktionen

DoExecute

```
int DoExecute();
```

Erzeugt das Objekt *TFileOpenDialog.*

Siehe auch: *TDialog::DoExecute*

Klasse TFileSaveDialog — opensave.h

TFileSaveDialog ist ein modaler Dialog, der Ihnen die Eingabe einer zu speichernden Datei ermöglicht. Verwenden Sie *TFileSaveDialog*, um auf einen CM_FILESAVEAS-Befehl zu antworten, der erzeugt wird, wenn ein Anwender Datei/Speichern aus einem Menü wählt. *TFileSaveDialog* verwendet die Struktur *TOpenSave::TData*, um das Datei-Speichern-Dialogfenster zu initialisieren.

Öffentliche Konstruktoren

Konstruktor

```
TFileSaveDialog(TWindow* parent, TData& data, TResID templateID = 0,
                const char far* title = 0, TModule* module = 0);
```

Konstruiert und initialisiert das *TFileOpen*-Objekt, das auf der Information in der Datenstruktur *TOpenSaveDialog::TData* basiert, die Angaben zum Dateinamen, dem Dateiverzeichnis und Dateinamen-Suchfiltern enthält.

Siehe auch Struktur *TOpenSaveDialog::TData, TModule, TResID, TWindow*

Öffentliche Elementfunktionen

DoExecute

```
int DoExecute();
```

Erzeugt das Objekt *TFileSaveDialog*.

Siehe auch: *TDialog::DoExecute, TOpenSaveDialog*

Klasse TFilterValidator — validate.h

Eine streamfähige Klasse mit Namen *TFilterValidator* überprüft ein Eingabefeld, während der Benutzer hineinschreibt. Das Prüfungsobjekt besitzt eine Anzahl erlaubter Zeichen. Wenn der Benutzer ein Zeichen eingibt,

zeigt die Filterprüfung an, ob das Zeichen gültig oder ungültig ist. Siehe *TValidator* für ein Beispiel zur Eingabeprüfung.

Öffentliche Konstruktoren

Konstruktor

```
TFilterValidator(const TCharSet& validChars);
```

Konstruiert ein Filterprüfungsobjekt, indem zuerst der von *TValidator* geerbte Konstruktor aufgerufen und dann *ValidChars* auf *validChars* gesetzt wird.

Öffentliche Elementfunktionen

Error

```
void Error();
```

Error überschreibt die virtuelle Funktion von *TValidator* und zeigt ein Meldungsfenster an. Damit wird angezeigt, daß der Text-String ein ungültiges Zeichen enthält.

Siehe auch: *TValidator::Error*

IsValid

```
bool IsValid(const char far* str);
```

IsValid überschreibt die virtuelle Funktion von *TValidator* und gibt TRUE zurück, wenn alle Zeichen in *str* in der Anzahl der erlaubten Zeichen, *Valid Char*, enthalten sind. Andernfalls gibt die Funktion FALSE zurück.

IsValidInput

```
bool IsValidInput(char far* str, bool suppressFill);
```

IsValidInput überschreibt die virtuelle Funktion von *TValidator* und prüft jedes Zeichen im String *str*, um sicherzustellen, das es in der Anzahl der erlaubten Zeichen, *ValidChar,* enthalten ist. *IsValidInput* gibt TRUE zurück, wenn alle Zeichen in *str* gültig sind. Andernfalls gibt die Funktion FALSE zurück.

Siehe auch: *TValidator::IsValidInput*

Geschützte Datenelemente

ValidChars

```
TCharSet ValidChars;
```

Enthält die Anzahl aller Zeichen, die der Benutzer schreiben kann. Um zum Beispiel nur numerische Ziffern zuzulassen, setzen Sie *ValidChars* auf "0-9". *ValidChars* wird durch den Parameter *validChars* gesetzt, der an den Konstruktor übergeben wurde.

Klasse TFindDialog — findrepl.h

TFindDialog-Objekte stellen nicht-modale Dialogfenster-Oberflächenelemente dar, die die Angabe des zu findenden Textes ermöglichen. *TFindDialog* kommuniziert mit dem besitzenden Fenster über eine registrierte

Botschaft. *TFindDialog* ist von *TFindReplaceDialog* abgeleitet und verwendet die Struktur *TFindReplaceDialog::TData*, um das Dialogfeld mit vom Anwender eingegebenen Werten (wie beispielsweise dem zu suchenden Text-String) zu initialisieren.

Öffentliche Konstruktoren

Konstruktor

```
TFindDialog(TWindow* parent, TData& data, TResId templateId = 0,
        const char far* title = 0, TModule* module = 0);
```

Konstruiert ein *TFindDialog*-Objekt mit dem gegebenen übergeordneten Fenster, der Ressourcen-ID und dem Titel. Setzt die Attribute des Dialogfensters, das auf der Information in der Struktur *TFindReplaceDialog::TData* basiert, die Angaben über den zu suchenden Text-String enthält.

Siehe auch: *TFindReplaceDialog::TData*

Geschützte Elementfunktionen

DoCreate

```
HWND DoCreate();
```

Erzeugt ein nicht-modales Oberflächenelement für einen Dialog zum Suchen.

Klasse TFindReplaceDialog — findrepl.h

TFindReplaceDialog ist eine abstrakte Basisklasse für ein nicht-modales Dialogfenster, das ein Suchen und Ersetzen von Text ermöglicht. Diese Basisklasse enthält Funktionen, die den beiden abgeleiteten Klassen *TFindDialog* und *TReplaceDialog* eigen sind. *TFindDialog* läßt Sie den zu suchenden Text angeben und *TReplaceDialog* den zu ersetzenden Text. *TFindReplaceDialog* kommuniziert mit dem besitzenden Fenster über eine registrierte Botschaft.

Öffentliche Konstruktoren

Konstruktor

```
TFindReplaceDialog(TWindow* parent, TData& data, TResId templateId = 0,
                   const char far* title = 0, TModule* module = 0);
```

Konstruiert ein *TFindReplaceDialog*-Objekt mit einem übergeordneten Fenster, einer Ressourcen-ID und einem Titel. Setzt die Attribute des Dialogfensters mit den angegebenen Daten aus der Struktur *TFindReplaceDialog::TData*.

Siehe auch: Struktur *TFindReplaceDialog::TData, TModule, TResID, TWindow*

Öffentliche Elementfunktionen

CmCancel

```
void Cm Cancel();
```

Antwortet auf ein Klicken auf den »Abbruch«-Schalter.

CmFindNext

```
void CmFindNext();
```

Antwortet auf ein Klicken auf den »Weitersuchen«-Schalter.

CmReplace

```
void CmReplace();
```

Antwortet auf ein Klicken auf den »Ersetzen«-Schalter.

CmReplaceAll

```
void CmReplaceAll();
```

Antwortet auf ein Klicken auf den »Alles ersetzen«-Schalter.

EvNCDestroy

```
void EvNCDestroy();
```

Ruft *TWindow::EvNCDestroy* auf, die auf eine eintreffende EV_WM_NCDESTROY-Botschaft antwortet, die dem besitzenden Fenster mitteilt, daß sein Nicht-Client-Bereich entfernt wird.

Siehe auch: *TWindow::EvNCDestroy*

Geschützte Datenelemente

Data

```
TData& Data;
```

Data ist eine Referenz auf das *TData*-Objekt, das im Konstruktor übergeben wurde.

Siehe auch: Klasse *TFindReplace::TData*

fr

```
FINDREPLACE fr;
```

Eine Struktur, die Attribute zum Suchen und Ersetzen enthält wie zum Beispiel die Größe des Suchen-Puffers und Zeiger auf Suche- und Ersetzen-Strings, die für Suchen-und-Ersetzen-Operationen verwendet werden.

Siehe auch Klasse FINDREPLACE

Geschützte Elementfunktionen

DoCreate

```
HWND DoCreate() = 0;
```

DoCreate ist eine virtuelle Funktion, die in abgeleiteten Klassen überschrieben wird, um ein nicht-modales Dialogfenster zum Suchen und Ersetzen zu erzeugen.

DialogFunction

```
bool DialogFunction(uint message, WPARAM, LPARAM);
```

Gibt TRUE zurück, wenn die Botschaft behandelt wird.

Siehe auch: *TDialog::DialogFunction*

Init

```
void Init(TResId templateId);
```

Init wird von Konstruktoren in abgeleiteten Klassen benutzt und initialisiert ein *TFindReplaceDialog*-Objekt mit der aktuellen Ressourcen-ID und anderen Elementen.

Anworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_WM_NCDESTROY	EvNCDestroy

Klasse TFindReplaceDialog::TData — findrepl.h

Die Klasse *TFindReplaceDialog::TData* definiert Informationen, die für die Initialisierung eines *TFindReplace*-Dialogfensters notwendig sind. Die Klassen *TFindDialog* und *TReplaceDialog* verwenden die Klasse *TFindReplaceDialog::TData*, um das Dialogfenster zu initialisieren und um vom Benutzer eingegebene Optionen wie beispielsweise den Suchtext und den zu ersetzenden Text zu erhalten.

Öffentlicher Konstruktor und Destruktor

Konstruktor

```
TFindReplaceDialog(uint32 flags = 0, int bufferSize = 81);
```

Konstruiert ein *TData*-Objekt mit dem angegebenen Flag-Wert, der den Status der Schalterelemente des Dialogfensters und die Puffergröße für die Suchen- und Ersetzen-Strings initialisiert.

Destruktor

```
~TData()
```

Entfernt ein ~*TData*-Objekt.

Öffentliche Datenelemente

BuffSize

```
int BuffSize;
```

BuffSize enthält die Größe des Textpuffers.

Error

```
uint32 Error;
```

Wird das Dialogfenster erfolgreich erzeugt, ist *Error* 0. Andernfalls enthält *Error* einen oder mehrere der folgenden Fehlercodes:

Konstante	Bedeutung
CDERR_LOCKRESOURCEFAILURE	Angegebene Ressource konnte nicht gesperrt werden.
CDERR_LOADRESFAILURE	Angegebene Ressource konnte nicht geladen werden.
CDERR_LOADSTRFAILURE	Angegebener String konnte nicht geladen werden.
CDERR_REGISTERMSGFAIL	Die Fensterbotschaft (ein Wert, der für die Kommunikation zwischen Anwendungen verwendet wird) kann nicht registriert werden. Dieser Botschaftswert wird verwendet, wenn Fensterbotschaften versandt oder hinterlegt werden.

FindWhat

```
char* FindWhat;
```

FindWhat enthält den Such-String.

Flags

```
uint32 Flags;
```

Flags zeigt den Zustand der Kontrollschalter und die im Dialogfenster aufgetretene Aktion an. Der Wert kann eine Kombination der folgenden Konstanten, die angeben, welchen Befehl der Anwender auswählen will, sein:

Konstante	Bedeutung
FR_DOWN	Der Nach-unten-Schalter in der Suchrichtung-Gruppe des Suchen-Dialogfensters ist ausgewählt.
FR_HIDEMATCHCASE	Das Markierungsfeld zur Beachtung der Groß-/Kleinschreibung ist versteckt.
FR_HIDEWHOLEWORD	Das Markierungsfeld für das Suchen ganzer Wörter ist versteckt.
FR_HIDEUPDOWN.	Die Schalter für die Suche Nach-oben und Nach-unten sind versteckt.
FR_MATCHCASE	Das Markierungsfeld zur Beachtung der Groß-/Kleinschreibung ist markiert.
FR_NOMATCHCASE	Das Markierungsfeld zur Beachtung der Groß-/Kleinschreibung ist deaktiviert. Diese Einstellung tritt bei der ersten Initialisierung des Dialogfensters auf.
FR_NOUPDOWN	Die Schalter für die Suche *Nach-oben* und *Nach-unten* sind deaktiviert. Diese Einstellung tritt bei der ersten Initialisierung des Dialogfensters auf.
FR_NOWHOLEWORD	Das Markierungsfeld für das Suchen ganzer Wörter ist deaktiviert. Diese Einstellung tritt bei der ersten Initialisierung des Dialogfensters auf.
FR_REPLACE	Der Schalter zum Ersetzen wurde im Ersetzen-Dialogfenster gedrückt.
FR_REPLACEALL	Der Schalter, um alles zu ersetzen, wurde im Ersetzen-Dialogfenster gedrückt.
FR_WHOLEWORD	Das Markierungsfeld für das Suchen ganzer Wörter ist markiert.

Replacewith

```
char* ReplaceWith;
```

ReplaceWith enthält den String zum Ersetzen.

Siehe auch: *TEditSearch::SearchData*, *TFindReplaceDialog::Data*

Klasse TFloatingFrame floatfra.h

Als eine von *TFrameWindow* und *TTinyCaption* abgeleitete Klasse implementiert *TFloatingFrame* einen fließenden Rahmen, der überall im übergeordneten Fenster positioniert werden kann. Mit der Ausnahme, daß eine schmale Titelleiste hinzugefügt wird, ist das Standardverhalten von *TFrameWindow* und *TFloatingFrame* dasselbe. Deshalb kann eine Anwendung, die *TFrameWindow* benutzt, auf einfache Weise die Funktionalität von *TFloatingFrame* erhalten, indem der Klassennamen einfach nur in *TFloatingFrame* geändert wird.

Wenn es ein Client-Fenster gibt, wird der fließende Rahmen für eine Anpassung an das Client-Fenster verkleinert. Dabei wird Raum für Ränder am oberen, unteren, linken und rechten Ende des Rahmens übriggelassen. Da der fließende Rahmen das Client-Fenster erwartet, daß das Client-Fenster seinen eigenen Hintergrund zeichnet, führt er in Antwort auf eine WM_ERASEBKGND-Botschaft nichts aus. Wenn allerdings kein Client-Fenster existiert, löscht der fließende Rahmen den Client-Hintergrundbereich über COLOR_BTNFACE.

Siehe PAINT.CPP, das Beispielprogramm auf Ihrer mitgelieferten Diskette, für ein Beispiel eines fließenden Rahmens.

Öffentliche Konstruktoren

Konstruktor

```
TFloatingFrame(TWindow *owner, char *title = 0, TWindow*
               clientWnd = 0, bool shrinkToClient = false, int
               CaptionHeight = DefaultCaptionHeight, bool popupPalette =
false, Module* module = 0);
```

Konstruiert ein *TFloatingFrame*-Objekt, das mit dem angegebenen übergeordneten Fenster verbunden ist. Standardmäßig wird das fließende Rahmenfenster nicht verkleinert, um sich dem Client-Fenster anzupassen, und die Eigenschaft der fließenden Palette ist nicht aktiviert.

Setzen Sie *enablePalette* auf TRUE, wenn Sie die Eigenschaft der fließenden Palette für das Fenster aktivieren wollen. Die fließende Palette ist ein Popup-Fenster mit einem kleinen Titel, einem Standardfensterrand und einem Schließsymbol an Stelle eines Systemmenüfeldes. Es gibt keine Vollbild- oder Symbol-Schalter. Ein Rand mit der Breite eines Pixel wird um den Client-Bereich für den Fall hinzugefügt, wenn eine Mauspalette implementiert ist.

Diese Eigenschaft muß angeschaltet werden, bevor das Fenster erzeugt wird. Nach der Erzeugung des Fensters kann dessen Eigenschaft nicht verändert werden.

Siehe auch: *TFrameWindow::TFrameWindow, TTinyCaption::TTinyCaption*

Öffentliche Elementfunktionen

SetMargins

```
void SetMargins(const TSize& margin);
```

Setzt die Ränder des fließenden Palettenfensters auf die in *margin* angegebene Größe und setzt die Höhe der kleinen Titelleiste.

Siehe auch: *TTinyCaption::EnableTinyCaption*

Geschützte Elementfunktionen

DoNCHitTest

```
TEvent Status DoNCHitTest(TPoint& screenPt,uint& evRes);
```

Wenn das fließende Palettenfenster nicht aktiviert ist, wird *esPartial* zurückgegeben. Andernfalls wird eine Botschaft an das fließende Palettenfenster geschickt, daß sich Maus oder Cursor bewegt haben, und es wird *esComplete* zurückgegeben.

Anworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_WM_SYSCOMMAND	EvSysCommand
EV_WM_NCCALCSIZE	EvNCCalcSize
EV_WM_NCPAINT	EvNCPaint
EV_WM_NCHITTEST	EvNcHitTest

Klasse TFont — gdiobjec.h

TFont ist von *TGdiObject* abgeleitet und stellt Konstruktoren für das Erzeugen von Schriftobjekten von einer expliziten Information aus oder auf indirekte Weise bereit.

Öffentliche Konstruktoren

Konstruktor

```
TFont(HFONT handle, TAutoDelete autoDelete = NoAutoDelete);
```

Erzeugt ein *TFont*-Objekt und setzt das Datenelement *Handle* auf das gegebene entliehene *handle*. Das Datenelement *ShouldDelete* liegt stan-

dardmäßig bei FALSE. Damit wird sichergestellt, daß das entliehene Handle nicht gelöscht wird, wenn das C++ Objekt entfernt wird.

Konstruktor

```
TFont(const char far* facename = 0, int height = 0, int width = 0,
      int escapement = 0, int orientation = 0, int weight = FW_NORMAL,
      uint8 pitchAndFamily = DEFAULT_PITCH|FF_DONTCARE, BYTE italic
      = false, BYTE underline = false, uint8 strikeout = false,
      uint8 charSet = 1, uint8 outputPrecision = OUT_DEFAULT_PRECIS,
      uint8 clipPrecision = CLIP_DEFAULT_PRECIS, uint8 quality
      = DEFAULT_QUALITY);
```

Erzeugt ein *TFont*-Objekt mit den gegebenen Werten.

Konstruktor

```
TFont(int height, int width, int escapement = 0, int orientation = 0, int
weight = FW_NORMAL, uint8 italic = false, BYTE underline = false, uint8
strikeout = false, uint8 charSet = 1, uint8 outputPrecision =
OUT_DEFAULT_PRECIS, uint8 clipPrecision = CLIP_DEFAULT_PRECIS, uint8
quality = DEFAULT_QUALITY uint pitchAndFamily =
DEFAULT_PITCH|FF_DONTCARE, const char far* facename = 0);
```

Erzeugt ein Font-Objekt mit den gegebenen Werten. Die Parameterliste und die Standardwerte des Konstruktors entsprechen dem Windows API Aufruf *CreateFont*.

Konstruktor

```
TFont(const LOGFONT far* logFont);
```

Erzeugt ein *TFont*-Objekt aus dem gegebenen *logFont*.

Konstruktor

```
TFont(const TFont& font);
```

Der Kopierkonstruktor *TFont*.

Siehe auch *::CreateFont* (Windows API), *::CreateFontIndirect* (Windows API), *TGdiObject::Handle, TGdiObject::ShouldDelete*

Öffentliche Elementfunktionen

GetObject

```
bool GetObject(LOGFONT far& logFont) const;
```

Ermittelt Informationen über dieses Stiftobjekt und plaziert sie in die gegebene Struktur *LOGFONT*. Gibt bei Erfolg TRUE zurück und bei Nichterfolg FALSE.

Siehe auch: *TGdiObject::GetObject*, Struktur *LOGFONT*

operator HFONT()

```
operator HFONT() const;
```

Typumwandlungoperator, der das *Handle* dieser Schrift in den Typ *HFONT* konvertiert (der Datentyp, der das Handle auf eine physikalische Schrift repräsentiert).

Klasse TFrameWindow framewin.h

TFrameWindow ist von *TWindow* abgeleitet und kontrolliert solch Fenster-spezifisches Verhalten, wie Tastatursteuerung und Kommandoverarbeitung für Client-Fenster. Wenn zum Beispiel ein Fenster reaktiviert wird, ist *TFrameWindow* für das Wiederherstellen des Eingabefokus eines Fensters und für das Hinzufügen der Menüleiste und der Symbol-Unterstützung verantwortlich. *TFrameWindow* ist eine streamfähige Klasse.

Man unterscheidet bei den Fensterbereichen den Rahmen- und den Client-Bereich. Der Rahmenbereich besteht aus der Umrandung, System-Menüs, Mauspaletten und Statuszeilen. Der Client-Bereich schließ diese Teile nicht mit ein. Obwohl Rahmenfenster ein Client-Fenster unterstützen können, bleibt das Rahmenfenster vom Client-Fenster getrennt, so daß das Client-Fenster gewechselt werden kann, ohne daß sich das Rahmenfenster ändert.

ObjectWindows verwendet diese Rahmen- und Client-Struktur sowohl für die *TFrameWindow*- wie auch für die *TMDIChild*-Klasse. Beide Klassen können eine Client-Klasse enthalten. Das Vorhandensein einer eigenen Klasse für den Client-Bereich des Fensters gibt Ihren Programmen mehr Flexibilität. Beispielsweise kann dieser getrennte Client-Bereich, der ein Dialogfenster sein kann, in ein anderes Rahmenfenster (entweder in ein Hauptfenster oder ein untergeordnetes MDI-Fenster) verschoben werden.

Siehe *TFloatingFrame* für eine Beschreibung eines fließenden Rahmens mit derselben Standardfunktionalität wie ein Rahmenfenster.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TFrameWindow(TWindow* parent, const char far *title = 0,
             TWindow *clientWnd = 0, bool shrinkToClient = false,
             TModule* module = 0);
```

Konstruiert ein Fensterobjekt mit dem in *parent* gelieferten übergeordneten Fenster, das Null ist, wenn es sich um das Hauptfenster handelt. *title*, voreingestellt Null, enthält den Titel, der in der Titelzeile des Fensters angezeigt wird. *clientWnd* ist das Client-Fenster für dieses Rahmenfenster oder Null, wenn ein solches nicht existiert. *shrinkToClient* steuert, ob sich das Client-Fenster in der Größe an den Rahmen anpaßt, oder ob das Rahmenfenster sich an den Client anpaßt. Bitte beachten Sie, daß dieser Parameter nur die Größe des Hauptfensters beeinflußt. Wird ein Client-Fenster in einem Rahmenfenster, in dem *shrinkToClient* nicht gesetzt ist, verwendet, ändert das Client-Fenster seine Größe so, daß es in das Rahmenfenster paßt. Wird ein Client-Fenster in einem Rahmenfenster, in dem *shrinkToClient* gesetzt ist, ver-

wendet, verkleinert das Rahmenfenster seine Größe so, daß es in das Client-Fenster paßt.

Konstruktor

```
TFrameWindow(HWND hWnd, TModule* module = 0);
```

Konstruktor für ein *TFrameWindow*, das als Alias für ein Nicht-ObjectWindows-Fenster benutzt wird. *hWnd* ist das Handle zu dem bestehenden Fensterobjekt, das von *TFrameWindow* gesteuert wird. *module* enthält das Modul, das an den Konstruktor der Basisklasse übergeben wird.

Destruktor

```
~TFrameWindow;
```

Löscht jeden assoziierten Menüdeskriptor.

Siehe auch: *TWindow::TWindow*, *TFloatingFrame::TFloatingFrame*

Öffentliche Datenelemente

KeyboardHandling

```
bool KeyboardHandling;
```

Zeigt an, ob Tastatursteuerung erforderlich ist.

Öffentliche Elementfunktionen

AssignMenu

```
virtual bool AssignMenu(TResId menuResId);
```

Setzt *Attr.Menu* auf den gelieferten Wert *menuResId* und gibt alle vorherigen Strings, auf die *Attr.Menu* zeigt, frei. Wenn *HWindow* ungleich Null ist, lädt und setzt die Funktion das Menü des Fensters. Dabei wird das vorher existierende Menü entfernt.

Siehe auch: *TMDIFrameWindow::AssignMenu*, *TFrameWindow::SetMenuDescr*

EnableKBHandler

```
void EnableKBHandler();
```

Setzt ein Flag, das anzeigt, daß der Empfänger die Tastatursteuerung angefordert hat (die Übersetzung von Tastatureingabe in Dialogelementauswahlen). Die Tastaturschnittstelle ermöglicht dem Benutzer die Verwendung der Tabulator- und Pfeiltasten, um sich zwischen den Dialogelementen zu bewegen. Sie ist standardmäßig für Fenster und Dialogfenster deaktiviert.

GetClientWindow

```
virtual TWindow *GetClientWindow();
```

Gibt einen Zeiger auf das Client-Fenster zurück. Wenn Sie versuchen, auf ein fensterbasiertes Objekt in einem TMDIChild (dies ist ein Rahmenfenster) zuzugreifen, können Sie diese Funktion verwenden.

Siehe auch: *TMDIChild*

GetCommandTarget

```
virtual HWND GetCommandTarget();
```

Lokalisiert das untergeordnete Fenster, das das Ziel der Befehls- und Befehlsaktivierungsbotschaften ist und gibt dies Fenster zurück. Besitzt die

aktuelle Anwendung nicht den Fokus oder liegt der Fokus in einer Mauspalette der Anwendung, gibt *GetCommandTarget* das zuletzt aktive untergeordnete Fenster zurück.

Die Hauptfensterklasse des Anwenders kann diese Funktion überschreiben, wenn eine andere Form der Befehlsverarbeitung gewünscht wird. Die *TFrameWindow*-Funktionen *EvCommand* und *EvCommandEnable* verwenden *GetCommandTarget*, um das Zielfenster des Befehls zu finden. Diese Elementfunktion steht unter Presentation Manager nicht zur Verfügung.

GetMenuDescr

```
const TMenuDescr* GetMenuDescr();
```

Gibt einen Zeiger auf einen Menüdeskriptor zurück.

Siehe auch: *TFrameWindow::SetMenuDescr, TMenuDescr*

HoldFocusHwnd

```
bool HoldFocusHWnd(HWND hWndLose, HWND hWndGain);
```

Überschreibt die virtuelle Funktion von *TWindow*. Antwortet auf eine Anforderung durch ein untergeordnetes Fenster, um dessen HWND zu halten, wenn es den Fokus verliert. Speichert den HWND des untergeordneten Fensters in *HwndRestoreFocus*.

Siehe auch: *TWindow::HoldFocusHwnd*

IdleAction

```
void IdleAction(long idleCount);
```

Überschreibt die virtuelle Funktion von *TWindow*. *TApplication* ruft *IdleAction* des Hauptfensters auf, wenn keine Botschaften auf ihre Verarbeitung warten. *TFrameWindow* benutzt diese Idle-Zeit, um die Kommandoaktivierung für die Menüleiste durchzuführen. Sie gibt ferner *IdleAction* an jedes ihrer untergeordneten Fenster weiter. *IdleAction* kann für Hintergrundverarbeitung überschrieben werden.

Siehe auch: *TApplication::IdleAction*

MergeMenu

```
bool MergeMenu(const TMenuDescr& childMenuDescr);
```

Vermischt den gegebenen Menüdeskriptor mit dem eigenen Menüdeskriptor des Rahmens und zeigt das resultierende Menü in diesem Rahmen an. Siehe *TMenuDescr* für eine Beschreibung von Menüleistentypen, die vermischt werden können.

Siehe auch: *TMenuDescr*

PreProcessMsg

```
bool PreProcessMsg(MSG& msg);
```

Überschreibt die virtuelle Funktion von *TWindow*. Führt die Vorverarbeitung von Fensterbotschaften durch. Wenn das untergeordnete Fenster die Tastatursteuerung angefordert hat, behandelt *PreProcessMsg* alle Tastenkürzelbotschaften und ruft dann die Windows-API- Funktion *::IsDialogMessage* auf, um alle anderen Tastaturbotschaften zu verarbeiten.

Siehe auch: *TWindow::PreProcessMsg*

RestoreMenu

```
bool Restore Menu();
```

Stellt das Standardmenü des Rahmenfensters wieder her.

SetClientWindow

```
virtual TWindow *SetClientWindow(TWindow* clientWindow);
```

Setzt das Client-Fenster auf das angegebene Fenster. Der Anwender ist dafür verantwortlich,das alte Client-Fenster zu entfernen, wenn sie dies löschen möchten.

SetDocTitle

```
bool SetDocTitle(const char far* docname, int index);
```

Überschreibt die virtuelle Funktion von *TWindow*. Fügt die Nummer der View in den Titel ein und zeigt dann diese Nummer auf dem Bildschirm an. Diese Funktion kann überschrieben werden, wenn Sie nicht die Standard-Implementierung, die die Nummer auf dem Bildschirm anzeigt, verwenden wollen, sondern beispielsweise anstelle von ":2" "Zwei" anzeigen möchten. Ein Beispiel für das Verhalten dieser Funktion, die die Views neu numeriert, wenn eine von ihnen geschlossen wird, finden Sie in Schritt 13 des ObjectWindows Tutorial.

SetIcon

```
bool SetIcon(TModule* iconModule, TResId iconResId);
```

Setzt das Symbol im Modul, das in *iconModule* angegeben wird, auf die in *iconResId* angegebene Ressourcen-ID. Ein Beispiel für das Zeichnen eines Symbols aus einer Bitmap finden Sie in der Beispieldatei bmpview.cpp. Sie können *iconResId* auf einen dieser vordefinierten oder auf benutzerdefinierte Werte setzen:

```
IDI_APPLICATION Standardsymbol für Anwendungen.
IDI_ASTERISK    Sternchen, das für eine Informationsbotschaft verwendet wird.
IDI_EXCLAMATION Ausrufezeichen für eine Warnmeldung.
IDI_HAND        Handsymbol für Warnmeldungen.
IDI_QUSTION     Fragezeichen zur Bestätigung einer Antwort.
```

Siehe auch: *TFrameWindow::EvQueryDragIcon*

SetMenu

```
virtual bool SetMenu(HMENU newMenu);
```

Überschreibt die nicht-virtuelle Funktion *SetMenu* von *TWindow*. Hierdurch wird es abgeleiteten Klassen ermöglicht, diese Funktion anders als *TWindow* zu implementieren. *SetMenu* setzt das Menü des Fensters auf das durch *newMenu* angegebene Menü. Wenn *newMenu* 0 ist, wird das aktuelle Menü des Fensters entfernt. *SetMenu* gibt 0 zurück, wenn das Menü unverändert bleibt. Andernfalls wird ein Wert ungleich Null zurückgegeben.

SetMenuDescr

```
void SetMenuDescr(const TMenuDescr& menuDescr);
```

Setzt den Menüdeskriptor auf den neuen Menüdeskriptor.

Siehe auch: *TFrameWindow::GetMenuDescr, TMenuDescr*

Geschützte Datenelemente

ClientWnd

```
TWindow *ClientWnd;
```

ClientWnd zeigt auf das Client-Fenster des Rahmens.

DocTitleIndex

```
int DocTitleIndex;
```

Hält die Indexnummer für den Dokumenttitel.

HWndRestoreFocus

```
HWND HWndRestoreFocus;
```

Speichert das Handle des untergeordneten Fensters, dessen Fokus wiederhergestellt wird.

Siehe auch: *TFrameWindow::HoldFocusHwnd*

MergeModule

```
TModule* MergeModule;
```

Informiert das Rahmenfenster, aus welchem Modul das Menü stammt. *TDecoratedFrame* erhält aus diesem Element die Menühinweise, die am unteren Bildschirmrand angezeigt werden. Dabei wird angenommen, daß die Menühinweise sich dort finden, woher auch das Menü stammt.

Geschützter Konstruktor

Konstruktor

```
TFrameWindow();
```

Geschützter Konstruktor, der in Verbindung mit der Funktion *Init* zur Initialisierung virtuell abgeleiteter Klassen verwendet wird.

Geschützte Elementfunktionen

EvCommand

```
LRESULT EvCommand(uint id, HWND hWndCtl, uint notifyCode);
```

Stellt eine gesonderte Verarbeitung für Kommandos bereit und überläßt die Behandlung des Kommandos zuerst dem Fokus-Fenster und seinem übergeordneten Fenster.

EvCommandEnable

```
void EvCommandEnable(TCommandEnabler& ce);
```

Behandelt das Markieren sowie das Aufheben der Markierung bei den Menüeinträgen des Rahmenfensters.

EvEraseBkgnd

```
bool EvEraseBkgnd(HDC);
```

EvEraseBkgnd löscht den Hintergrund des in *HDC* angegebenen Fensters. Sie gibt TRUE zurück, wenn der Hintergrund gelöscht ist. Andernfalls gibt die Funktion FALSE zurück.

EvInitMenuPopup

```
HANDLE EvInitMenuPopup(HMENU hPopupMenu, uint index, bool sysMenu);
```

Wird *EvInitMenuPopup* vor der Anzeige eines Popup-Menüs gesendet, ermöglicht die Funktion der Anwendung, die Menüeinträge zu ändern, bevor

das Menü angezeigt wird. *EvInitMenuPopup* kontrolliert, ob die Einträge im Popup-Menü aktiviert oder deaktiviert, markiert oder nicht markiert oder Strings sind. *HMENU* zeigt das Menü-Handle an. *index* ist der Index des Popup-Menüs. *sysMenu* zeigt an, ob das Popup-Menü das Systemmenü ist.

EvPaint

```
void EvPaint();
```

Antwortet auf eine WM_PAINT-Botschaft, um das Symbol für Symbol Fenster zu zeichnen oder um Client-Fenstern eine Möglichkeit zu geben, das Symbol zu zeichnen.

Siehe auch: *TWindow::Paint*, *TScroller::BeginView*, *TScroller::EndView*

EvParentNotify

```
void EvParentNotify(uint event, uint childHandleOrX, uint childIDOrY);
```

Antwortet auf eine Botschaft zur Benachrichtigung des übergeordneten Fensters, daß ein gegebenes Ereignis aufgetreten ist. Wenn das Client-Fenster entfernt ist, schließt die Funktion das übergeordnete Fenster. Wenn *shrinkToClient* gesetzt ist, und sich die Größe des untergeordneten Fensters verändert hat, wird der Rahmen angepaßt.

Wenn ein Client-Fenster von *TFrameWindow* entfernt wird, bemerkt das Objekt *TFrameWindow* die Botschaft WM_PARENTNOTIFY und hinterlegt für sich selbst eine Botschaft zum Schließen. Gäbe es diese Botschaft nicht, bliebe ein leerer Rahmen zurück und das Client-Fenster müßte entscheiden, wie der Rahmen zu entfernen ist. Wenn Sie dies nicht wünschen, können Sie aus dem Rahmenfenster ableiten und Ihre Anwendung die Botschaften *EvParentNotify* oder *EvClose* verwalten lassen.

EvQueryDragIcon

```
HANDLE EvQueryDragIcon();
```

Antwortet auf eine WM_QUERYDRAGICON-Botschaft, die an ein als Symbol dargestelltes Fenster, das gerade gezogen wird, geschickt wird. An Stelle des Standardsymbols verwendet *EvQueryDragIcon* das Symbol, das über *SetIcon* gesetzt wurde.

Dieses Element steht unter Presentation Manager nicht zur Verfügung.

Siehe auch: *TFrameWindow::SetIcon*

EvSetFocus

```
void EvSetFocus(HWND hWndLostFocus);
```

Stellt den Fokus für das aktive Fenster wieder her. *hWndLostFocus* enthält das Handle auf das Fenster, das den Fokus verloren hat.

EvSize

```
void EvSize(uint sizeType, TSize &size);
```

Verändert die Größe des Client-Fensters wieder, so daß es äquivalent mit des Größe des Client-Rechteckes ist. Ruft *TWindow::EvSize* als Antwort auf eine eintreffende WM_SIZE-Botschaft auf.

Siehe auch: *TSize*

Init

```
void Init(TWindow* clientWnd, bool shrinkToClient);
```

Diese Initialisierungsfunktion wird für virtuell abgeleitete Klassen verwendet, die *Init* aufrufen müssen, bevor die Erzeugung abgeschlossen ist. Diese Prozedur stellt virtuell abgeleiteten Klassen die benötigten Daten zur Verfügung und sorgt dafür, daß die Daten in der richtigen Reihenfolge verfügbar gemacht werden.

SetupWindow

```
void SetupWindow();
```

Ruft *Twindow::SetUpWindow* auf, um Fenster in einer Liste untergeordneter Fenster zu erzeugen. *SetupWindow* führt die anfängliche Anpassung des Client-Fensters (wenn ein solches existiert) durch, setzt mit Hilfe des Menüdeskriptors das Menü des Rahmens und initialisiert *HwndRestoreFocus*.

Siehe auch: *TWindow::SetUpWindow*

Anworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_WM_ERASEBKGND	EvEraseBkgnd
EV_WM_INITMENUPOPUP	EvWmInitMenuPopup
EV_WM_PAINT	EvPaint
EV_WM_PARENTNOTIFY	EvParentNotify
EV_WM_QUERYDRAGICON	EvQueryDragIcon
EV_WM_SETFOCUS	EvWmSetFocus
EV_WM_SIZE	EvWmSize

Klasse TGadget gadget.h

TGadget ist die Basisklasse für folgende abgeleitete Gadget-Klassen.

Klasse	Beschreibung
TBitmapGadget	Zeigt eine Bitmap an.
TButtonGadget	Verwendet eine Bitmap, um ein Schalter-Gadget nachzuahmen.
TControlGadget	Kapselt das Einfügen eines Dialogelements - wie eines Editierfeldes oder eines Kombinationsfensters - in ein Gadget-Fenster.
TTextGadget	Zeigt Text an.
TSeparatorGadget	Unterteilt logische Gruppen von Gadgets.

TGadget-Oberflächenobjekte gehören zu einem Gadget-Fenster, besitzen Ränder und Rahmen sowie ein eigenes Koordinatensystem. Die Rahmen sind die gleichen wie für *TGadgetWindow*. Ränder werden immer in Randeinheiten (border units) bemessen.

Um die Attribute für das Gadget zu setzen, wird entweder ein Randstil ausgewählt (der automatisch die einzelnen Rahmenecken setzt), oder die Ränder werden gesetzt, und die Elementfunktion *PaintBorder* wird anschließend überschrieben, um das gewünschte Aussehen des Gadget zu erzielen. Durch Ändern der Ränder, der Rahmen oder des Rahmenstils wird die Elementfunktion *GadgetChangedSize* des Gadgetfensters aktiviert.

Obwohl standardmäßig eine automatische Größenanpassung der Gadgets an ihre Inhalte erfolgt, kann dieses Attribut durch Setzen eigener Werte für *ShrinkWrapWidth* und *ShrinkWrapHeight* gesteuert werden.

Als aktives Fenster empfängt ein Gadget-Fenster Botschaften von der Maus. Das Gadget-Fenster entscheidet nach dem Empfang der Botschaft, welchem Gadget sie zugeteilt werden soll und ruft dazu direkt die Elementfunktion auf, statt eine Botschaft zu verschicken oder zu hinterlegen.

Siehe auch: *TBitmapGadget, TButtonGadget, TControlGadget, TTextGadget, TSeparatorGadget*

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TGadget (int id = 0, TBorderStyle = None);
```

Erzeugt ein *TGadget*-Objekt mit den Angaben für ID und Rahmenstil.

Destruktor

```
virtual ~TGadget();
```

Zerstört ein *TGadget*-Oberflächenobjekt und entfernt es aus dem zugeordneten Fenster.

Öffentliche Datenelemente

Clip

```
bool Clip;
```

Wenn *Clip* **false** ist, wurden keine Clipping-Ränder festgelegt. Wenn *Clip* **true** ist, wird das Zeichnen für jedes Gadget durch den Gadgetrand begrenzt.

WideAsPossible

```
bool WideAsPossible;
```

WideAsPossible legt fest, ob die Gadgetbreite vom Gadgetfenster so groß wie im verfügbaren Platz möglich gesetzt wird. Die Standardeinstellung ist **false**.

Siehe auch: *TGadgetWindow::WideAsPossible*

Öffentliche Aufzählungstypen und Strukturen

TBorders

```
struct TBorders
   unsigned Left;
   unsigned Right;
```

```
unsigned Top;
unsigned Bottom;
```

Die Struktur *TBorders* enthält die Werte für die linken, rechten, oberen und unteren Abmessungen des Gadget.

TBorderStyle

```
enum TBorderStyle {None, Plain, Raised, Recessed, Embossed};
```

Zählt eine exklusive Liste von Rahmenstilen auf: none, plain, raised (das Gadget scheint dreidimensional aus dem Gadget-Fenster hervorzustehen), recessed (das Gadget erscheint dreidimensional vertieft im Gadget-Fenster) oder embossed (das Gadget hat einen geprägt wirkenden Rahmen). Einen Eindruck von möglichen Rahmenstilen gibt das ObjectWindows-Beispielprogramm MDIFILE.CPP auf der mitgelieferten Diskette.

TMargins

```
struct TMargins {
enum TUnits {Pixels, LayoutUnits, BorderUnits};
TUnits  Units;
int  Left;
int  Right;
int  Top;
int  Bottom;

};
```

TMargins enthält die Rahmenabmessungen für das Gadget und wird von *TGadgetWindow* und *TGadget*-Klassen benötigt. Der Konstruktor initialisiert *Units* auf LayoutUnits und setzt die Werte für *Left, Right, Top* und *Bottom* auf 0.

Siehe auch: *TGadgetWindow::Set Margins*

Öffentliche Elementfunktionen

CommandEnable

```
virtual void CommandEnable();
```

CommandEnable steht zur Verfügung, damit das Gadget Befehle verarbeiten kann (um beispielsweise auf eingehende Botschaften zu reagieren, wenn es sinnvoll ist).

GetBorders

```
TBorders &GetBorders();
```

Holt die Rahmen für das Gadget. Rahmen werden in Rahmeneinheiten gemessen, die auf SM_CXBORDER und SM_CYBORDER basieren.

Siehe auch: *TGadget::Set Borders*

GetBorderStyle

```
TBorderStyle GetBorderStyle();
```

Holt den Rahmenstil des Gadgets.

Siehe auch: *TGadget::SetBorderStyle*

GetBounds

```
TRect &GetBounds();
```

Liefert das begrenzende Rechteck für das Gadget zurück.

Siehe auch: *TButtonGadget::SetNotchCorners*

GetDesiredSize

```
virtual void GetDesiredSize(TSize& size);
```

GetDesiredSize bestimmt die Größe des Gadgets. Das Gadgetfenster sendet diese Botschaft, um die Gadgetgröße abzufragen. Wenn automatische Anpassung angefordert wurde, liefert *GetDesiredSize* die für die Plazierung der Ränder und Rahmen erforderliche Größe zurück. Wurde keine automatische Größenanpassung angefordert, liefert es die aktuelle Breite und Höhe zurück. *TGadgetWindow* benötigt diese Information zur Bestimmung der Gadgetgröße. Es kann diese Abmessungen aber anpassen, wenn es erforderlich ist. Ist *WideAsPossible* **true**, wird der Breitenparameter (*size.cx*) ignoriert.

GetEnabled

```
bool GetEnabled();
```

Legt fest, ob das angegebene Gadget für Tastatur- und Mauseingaben aktiviert ist. Ist es aktiviert, liefert *GetEnabled* den Wert **true** zurück, anderenfalls **false**. Standardmäßig ist das Gadget für Tastatur- und Mauseingaben aktiviert.

Siehe auch: *TGadget::SetEnabled*

GetId

```
int GetId();
```

Holt die ID für das Gadget.

GetMargins

```
TMargins &GetMargins();
```

Holt die Randabmessungen.

GetOuterSizes

```
void GetOuterSizes(int& left, int& right, int& top, int& bottom);
```

Liefert den von Rändern und Rahmen eingenommenen Platz zurück (in Pixeln).

NextGadget

```
TGadget* NextGadget();
```

Liefert das nächste Gadget aus der Gadgetliste.

SetBorders

```
void SetBorders(TBorders& borders);
```

Setzt die Rahmen für das Gadget. Wenn die Rahmen geändert werden, ruft *SetBorders TGadgetWindow::GadgetChangedSize* auf, um das Gadgetfenster über die Änderung zu benachrichtigen.

Siehe auch: *TGadget::GetBorders, TGadgetWindow::GadgetChangedSize*

SetBorderStyle

```
void SetBorderStyle(TBorderStyle);
```

Bestimmt den Rahmenstil für das Gadget.

Siehe auch: *TGadget::GetBorderStyle*

SetBounds

```
virtual void SetBounds(TRect& rect);
```

SetBounds informiert das Gadget über eine Änderung seines begrenzenden Rechtecks. Das Standardverhalten bewirkt lediglich eine Aktualisierung der Instanzvariablen *Bounds*, doch kann diese Methode umgangen werden, um auch den internen Zustand des Gadget zu aktualisieren.

Set Enabled

```
virtual void SetEnabled(Bool);
```

Aktiviert oder deaktiviert die Tastatur- und Mauseingabe für das Gadget. Bei der Erzeugung ist das Gadget standardmäßig deaktiviert und muß vor der Benutzung aktiviert werden.

Siehe auch: *TGadget::GetEnabled*

SetMargins

```
void SetMargins(TMargins& margins);
```

Setzt die Ränder des Gadgets. Werden die Ränder geändert, ruft *SetMargins TGadgetWindow::GadgetChangedSize* auf, um das Gadgetfenster zu benachrichtigen.

Siehe auch: *TGadget::GetMargins*

SetShrinkWrap

```
void SetShrinkWrap(bool shrinkWrapWidth, bool shrinkWrapHeight);
```

Setzt die *ShrinkWrapWidth*- und *ShrinkWrapHeight*-Datenelemente. Die abgeleitete Klasse kann die *GadgetChangedSize*-Elementfunktion von *TGadgetWindow* aufrufen, wenn die Gadgetgröße geändert werden soll.

Siehe auch: *TGadgetWindow::GadgetChangedSize*

SetSize

```
void SetSize(TSize& size);
```

SetSize ändert die Größe des Gadget und ruft dann *TGadgetWindow::GadgetChangedSize* auf, um die Größenänderung durchzuführen.

Diese Funktion wird nur benötigt, wenn die automatische Rahmenanpassung in einer oder in beiden Dimensionen ausgeschaltet wurde; andernfalls ist die *GetDesiredSize*-Elementfunktion zu verwenden, um die automatisch angepaßte Größe zurückzuliefern.

SysColorChange

```
virtual void SysColorChange();
```

SysColorChange wird nach einem Wechsel der Systemfarben aufgerufen, um die Gadgets und ihre Farben neu aufzubauen, falls dies erforderlich ist.

Geschützte Datenelemente

Bounds

```
TRect Bounds;
```

Enthält das begrenzende Rechteck für das Gadgetfenster in Gadgetfenster-Koordinaten.

Siehe auch: *TGadget::GetInnerRect*

Borders

```
TBorders Borders;
```

Enthält die Rahmenabmessungen für das Gadget.

Siehe auch: *TGadget::GetInnerRect*

BorderStyle

```
TBorderStyle BorderStyle;
```

Enthält den Rahmenstil für das Gadget.

Id

```
int ID;
```

Enthält die Kennung des Gadgets.

Margins

```
TMargins Margins;
```

Enthält die Randabmessungen des Rechtecks.

Siehe auch: *TGadget::GetInnerRect*

ShrinkWrapHeight

```
bool ShrinkWrapHeight;
```

Legt die automatische Größenanpassung der Gadgetbegrenzungen an den Gadgetinhalt fest.

ShrinkWrapWidth

```
bool ShrinkWrapWidth;
```

Legt die automatische Größenanpassung der Gadgetbegrenzungen an den Gadgetinhalt fest.

TrackMouse

```
bool TrackMouse;
```

Als **false** initialisiert. Wenn *TrackMouse* **true** ist, hängt sich das Gadget bei *LButtonDown* an die Maus an und läßt bei *LButtonUp* wieder los. Dies geschieht durch Aufruf von *GadgetSetCapture* und *GadgetReleaseCapture* von *TGadgetWindow*.

Siehe auch: *TGadget::LButtonDown, TGadget::LButtonUp*

Window

```
TGadgetWindow* Window;
```

Referenziert das besitzende oder das übergeordnete Fenster für das Gadget.

Geschützte Elementfunktionen

GetInnerRect

```
void GetInnerRect(TRect& rect);
```

Berechnet den Flächeninhalt des Gadgetrechtecks ausschließlich der Ränder und Rahmen.

Inserted

```
virtual void Inserted();
```

Wird aufgerufen, nachdem ein Gadget in ein Fenster eingefügt wurde.

Invalidate

```
void Invalidate(bool erase = true);
```

Wird benutzt, um den aktiven Teil des Gadget (normalerweise der Nicht-Rahmenteil) für ungültig zu erklären. *Invalidate* ruft *InvalidateRect* auf und übergibt die Begrenzungsbreite und -höhe des zu löschenden Bereichs.

InvalidateRect

```
void InvalidateRect(const TRect& rect, bool erase = true);
```

Erklärt das gadgetbezogene Rechteck im übergeordneten Fenster für ungültig.

LButtonDown

```
virtual void LButtonDown(uint modKeys, TPoint& point);
```

Bewirkt das Anklinken an die Maus bei gesetztem *TrackMouse*. *point* befindet sich innerhalb des Koordinatensystems des Gadget.

Siehe auch: *TGadget::TrackMouse*

LButtonUp

```
virtual void LButtonUp(uint modKeys, TPoint& point);
```

Bewirkt das Loslassen der Maus bei gesetztem *TrackMouse*. *point* befindet sich innerhalb des Koordinatensystems des Gadget.

Siehe auch: *TGadget::TrackMouse*

MouseEnter

```
virtual void MouseEnter(uint modKeys, TPoint& point);
```

Wird beim Eintritt des Mauszeigers in das Gadget aufgerufen.

Siehe auch: *TGadget::MouseLeave*

MouseLeave

```
virtual void MouseLeave(uint modKeys, TPoint& point);
```

Wird aufgerufen, wenn der Mauszeiger das Gadget verläßt.

Siehe auch: *TGadget::MouseEnter*

MouseMove

```
virtual void MouseMove(uint modKeys, TPoint& point);
```

Wenn Mausereignisse aufgefangen werden, reagiert *EvMouseMove* auf eine Ziehbotschaft der Maus. *point* liegt im Koordinatensystem des Empfängers.

Siehe auch: *TGadget::MouseEnter*, *TGadget::MouseLeave*

Paint

```
virtual void Paint(TDC&);
```

Ruft *PaintBorder* auf, um das bezeichnete Geräteumfeld zu färben.

Siehe: *TTextGadget::Paint*

PaintBorder

```
virtual void PaintBorder(TDC& dc);
```

PaintBorder wird zum Einfärben des Rahmens verwendet und ruft *::GetSystemMetrics* auf, um die Breite und Höhe des Gadget zu erhalten. Die von *GetSysColor* zurückgelieferte Farbe wird zum Ausmalen oder Hervorheben der Fläche mit dem ausgewählten Pinsel benutzt. Abhängig vom gewählten Rahmenstil - herausgehoben, geprägt oder zurückgesetzt - färbt *PaintBorder* die angegebene Begrenzung. Für die Verwendung von

Rahmenstilen, die nicht von den ObjectWindows-Gadgets unterstützt werden, kann die Funktion umgangen werden.

PtIn

```
virtual bool PtIn(TPoint& point);
```

PtIn stellt fest, ob sich ein Punkt innerhalb des begrenzenden Rechtecks des Empfängers befindet und liefert, falls dies zutrifft, **true** zurück, andernfalls **false**.

Removed

```
virtual void Removed();
```

Wird aufgerufen, nachdem ein Gadget aus einem Fenster entfernt wurde.

Update

```
void Update();
```

Zeichnet das Gadget neu, sofern dies möglich ist.

Klasse TGadgetWindowFont gadgetwi.h

TGadgetWindowFont, abgeleitet aus *TFont*, ist eine spezifische Schriftart, die in Gadgetfenstern für Standardtext und zur Größenbestimmung verwendet wird. Die Schriftgröße in Punkt kann festgelegt werden (jedoch nicht die Größe in Pixeln), und die Schriftstile fett oder kursiv. Sie können eine der folgenden FF_xxxx-Konstanten zum Setzen der Schriftartfamilie verwenden:

Wert	Bedeutung
FF_DECORATIVE	Spezielle Schriftarten wie z.B. Old English.
FF_DONTCARE	Die Schriftart ist unwichtig.
FF_MODERN	Äquidistante Schriften mit konstanter Strichstärke, mit oder ohne Serifen, wie z.B. Pica, Elite oder Courier.
FF_ROMAN	Schriften mit variabler Strichstärke und Serifen, wie Times New Roman und New Century Schoolbook.
FF_SCRIPT	Schriften, die Handschrift ähneln, wie etwa Script.
FF_SWISS	Serifenlose Schriften mit variabler Strichstärke, wie MS Sans Serif.

Je nach verwendeter Schriftart kann die Strichstärke eine der folgenden Konstanten sein:

Wert	Bedeutung
FW_DONTCARE	Die Strichstärke ist unwichtig.
FW_THIN	Dünn
FW_EXTRALIGHT	Extraleicht
FW_ULTRALIGHT	Extraleicht
FW_LIGHT	Leicht
FW_NORMAL	Normal
FW_REGULAR	Normale Strichstärke

FW_MEDIUM	Mittlere Strichstärke
FW_SEMIBOLD	Halbfett
FW_DEMIBOLD	Halbfett
FW_BOLD	Fett
FW_EXTRBOLD	Extrafett
FW_ULTRBOLD	Extrafett
FW_BLACK	Sehr dicke Schriftstärke
FW_HEAVY	Sehr dicke Schriftstärke

Das Aussehen der Schriften hängt von der gewählten Schriftart ab. Bei einigen Schriften stehen nur FW_NORMAL, FW_REGULAR und FW_BOLD zur Verfügung. Ist FW_DONTCARE angezeigt, wird die Standardstrichstärke der Schrift verwendet.

Öffentlicher Konstruktor

Konstruktor

```
TGadgetWindowFont(int pointSize = 10, bool bold = false, bool italic =
false);
```

Erzeugt ein Oberflächenobjekt *TGadgetWindowFont* mit einer vorgegebenen Punktgröße von 10 Pica ohne die Attribute fett oder kursiv. Standardmäßig generiert der Konstruktor die Systemschrift, eine serifenlose Helvetica-Proportionalschrift.

Siehe auch: TFont

Klasse TGadgetWindow gadgetwi.h

TGadgetWindow - von *TWindow* abgeleitet - enthält eine Liste von nebeneinander angeordneten Gadgets für ein Fenster und gestattet die dynamische Anordnung von Mauspaletten. Folgende Attribute dieser Gadgets können festgelegt werden:

- Horizontale oder vertikale Anordnung. Plaziert die Gadgets horizontal oder vertikal innerhalb des inneren Rechtecks (dem Bereich ohne die Rahmen und Ränder).
- Schriftart. Standardschrift zur Verwendung in Gadgets und zur Berechnung der Layout-Einheiten. Weitere Informationen zu den Schriften finden Sie unter der Beschreibung von *TGadgetFont.*
- Linker, rechter, oberer und unterer Rand. Anzugeben in Pixeln, Layout-Einheiten (basierend auf der im Fenster verwendeten Schrift) oder Rahmeneinheiten (die Breite oder Höhe einer dünnen Fensterumrahmung).
- Maßeinheiten. Anzugeben in Pixeln, Layout-Einheiten oder Rahmeneinheiten.

- Größe des Gadgetfensters. Ein Gadgetfenster kann sich in Höhe, Breite oder in beiden Dimensionen automatisch an die Gadgetgröße anpassen. Voreingestellt ist die automatische Höhenanpassung für horizontal angeordnete Gadgets und die Breitenanpassung für vertikale Anordnung.

TGadgetWindow ist die Basisklasse für die folgenden abgeleiteten Klassen: *TControlBar, TMessageBar, TToolBox* und *TStatusBar*.

Öffentlicher Konstruktor und Destruktor

Konstruktor

```
TGadgetWindow(TWindow* parent = 0, TTileDirection direction = Horizontal,
              TFont *font = new TGadgetWindowFont, TModule* module = 0);
```

Erzeugt ein Oberflächenobjekt *TGadgetWindow* mit standardmäßiger Ausrichtung und Schriftart und übergibt *module* mit einem Standardwert von 0.

Destruktor

```
~TGadgetWindow();
```

Entfernt das Objekt *TGadgetWindow* durch Löschen aller seiner Gadgets und Fonts.

Typdefinitionen

THintMode

```
enum THintMode{NoHints, PressHints, EnterHints};
```

Zählt die Einstellungen für den Hinweismodus des Gadgets auf - entweder keine Hinweise, Hinweis beim Drücken eines Schalters oder wenn die Maus den Bereich eines Gadgets überstreicht.

Siehe auch: TGadgetWindow::GetHintMode

Öffentliche Elementfunktionen

FirstGadget

```
TGadget* FirstGadget() const;
```

Gibt das *FirstGadget* in der Liste zurück.

Siehe auch: *TGadgetWindow::FirstGadget*

GadgetChangedSize

```
void GadgetChangedSize(TGadget& gadget);
```

Wird benutzt, um dem Gadgetfenster mitzuteilen, daß ein Gadget seine Größe verändert hat. *GadgetChangedSize* ruft *LayoutSession* auf, um das Layout aller Gadgets neu vorzunehmen.

Siehe auch: *TGadget::SetShrinkWrap, TGadgetWindow::GadgetChangedSize*

GadgetFromPoint

```
TGadget* GadgetFromPoint(TPoint& point);
```

Liefert das Gadget an die angegebenen Fensterkoordinaten zurück.

GadgetReleaseCapture

```
void GadgetReleaseCapture(TGadget& gadget);
```

Hebt das Abfangen der Mausbotschaften auf, so daß sie von anderen Fenstern empfangen werden können.

Siehe auch: *TGadgetWindow::GadgetSetCapture*

GadgetSetCapture

```
bool GadgetSetCapture(TGadget& gadget);
```

GadgetSetCapture reserviert alle Mausbotschaften für das Gadgetfenster, bis das Abfangen wieder aufgehoben wird. Obwohl Gadgets immer informiert werden, wenn ein Mausereignis Linke-Maustaste-Drücken innerhalb des Rechtecks auftritt, muß die abgeleitete Gadgetklasse *GadgetSetCapture* aufrufen, wenn dem Gadget ein Mauszieh- oder ein Maustaste-Loslassen-Ereignis gemeldet werden soll.

Siehe auch: *TGadgetWindow::GadgetReleaseCapture*

GadgetWithId

```
TGadget* GadgetWithId(int id) const;
```

Liefert einen Zeiger auf das mit der gegebenen ID (*id*) assoziierte Gadget zurück.

GetDirection

```
TTileDirection GetDirection() const;
```

Holt die horizontale oder vertikale Ausrichtung der Gadgets.

Siehe auch: *TGadgetWindow::SetDirection*

GetFont

```
TFont &GetFont();
```

Gibt die Schriftart zurück (Voreinstellung ist *MS Sans Serif*).

Siehe auch: *TGadgetWindowFont::TGadgetWindowFont*

GetFontHeight

```
uint GetFontHeight() const;
```

Holt die Höhe der im Fenster verwendeten Schrift.

GetHintMode

```
THintMode GetHintMode();
```

Liefert den Hinweis-Modus zurück.

IdleAction

```
void IdleAction();
```

Solange keine Meldungen zur Verarbeitung anstehen, wird *IdleAction* aufgerufen und iteriert durch die Gadgets. Es ruft dabei ihre *CommandEnable* Elementfunktion auf.

Siehe auch: *TGadget::CommandEnable*

Insert

```
virtual void Insert(TGadget& gadget, TPlacement = After,
                    TGadget *sibling = 0);
```

Fügt ein Gadget vor oder hinter einem gleichgeordneten Gadget ein (*TPlacement*). Wenn *sibling* gleich 0 ist, wird das neue Gadget entweder am Anfang oder am Ende der Gadgetliste eingefügt. Wenn dieses Fenster bereits generiert worden ist, muß nach dem Einfügen von Gadgets *LayoutSession* aufgerufen werden.

Siehe auch. *TGadgetWindow::LayoutSession, TGadgetWindow::Remove*

LayoutSession

```
virtual void LayoutSession();
```

LayoutSession wird typischerweise dann aufgerufen, wenn sich die Größe der Gadgets oder ihrer Rahmen ändert, oder wenn Gadgets hinzugefügt oder entfernt werden. *LayoutSession* ruft *TileGadgets* zur Ausrichtung der Gadgets in der angegebenen Orientierung auf sowie *Invalidate*, um den Bereich als ungültig zu markieren (benötigt Neuzeichnung).

Siehe auch: *TGadgetWindow::Insert, TGadgetWindow::Remove, TWindow::Invalidate*

NextGadget

```
TGadget* NextGadget(TGadget& gadget) const;
```

Liefert das nächste Gadget nach *gadget* oder 0, wenn kein Gadget vorhanden ist.

Remove

```
virtual TGadget* Remove(TGadget& gadget);
```

Entfernt ein Gadget aus dem Gadgetfenster. Das Gadget wird zurückgeliefert, aber nicht zerstört. *Remove* liefert 0 zurück, wenn das Gadget nicht im Fenster ist.

Wenn dieses Fenster schon generiert worden ist, muß die aufrufende Anwendung *LayoutSession* aufrufen, nachdem irgendeins der Gadgets entfernt worden ist.

Siehe auch: *TGadgetWindow::Insert, TGadgetWindow::LayoutSession*

SetDirection

```
virtual void SetDirection(TTileDirection direction);
```

Bestimmt die horizontale oder vertikale Ausrichtung der Gadgets. Wenn das Gadgetfenster bereits erzeugt worden ist, paßt *SetDirection* die Abmessungen des Gadgetfensters an die Größe der Gadgets an.

Der Wert des Parameters *direction* hängt außerdem mit dem Wert des zweiten Parameters (*Tlocation*) in der Funktion *Insert* von *TDecoratedFrame* zusammen. *Tlocation* legt fest, an welcher Stelle die Dekoration in Relation zum Client-Fenster des Rahmenfensters eingefügt wird. Ist der zweite Parameter in *TDecoratedFrame::Insert* auf unten oder oben gesetzt, muß der Parameter *direction* in *SetDirection* auf horizontal gesetzt sein. Ist der zweite Parameter in *TDecoratedFrame::Insert* auf links oder rechts gesetzt, muß *direction* in *SetDirection* vertikal sein.

Siehe auch: *TGadgetWindow::GetDirection*

SetHintCommand

```
void SetHintCommand(int id);
```

Simuliert Menüauswahl-Botschaften, so daß die Befehlsverarbeitung von ObjectWindows Befehlshinweise anzeigen kann.

Siehe auch: WM_MENUSELECT, WM_ENTERIDLE

SetHintMode

```
void SetHintMode(THintMode hintMode);
```

Bestimmt den Anzeigemodus des Hinweistextes. Die Voreinstellung ist *PressHints* (zeigt Hinweistexte an, wenn ein Schalter gedrückt wird).

Siehe auch: *THintMode* enum

SetMargins

```
void SetMargins(TMargins &margins);
```

Setzt oder ändert die Ränder für das Gadgetfenster und ruft *LayoutSession* auf.

Siehe auch: *TGadgetWindow::Margins*

SetShrinkWrap

```
void SetShrinkWrap(bool shrinkWrapWidth, bool shrinkWrapHeight);
```

Setzt die Breite und Höhe der Datenelemente. Standardmäßig ist für horizontale Ausrichtung *ShrinkWrapWidth* **false** und *ShrinkWrapHeight* **true**. Analog dazu ist *ShrinkWrapWidth* bei vertikaler Ausrichtung standardmäßig **true** und *ShrinkWrapHeight* **false**.

Geschützte Datenelemente

AtMouse

```
TGadget* AtMouse;
```

Das letzte Gadget an der Mausposition.

BkgndBrush

```
TBrush* BkgndBrush;
```

Die Farbe des Hintergrundpinsels.

Capture

```
TGadget *Capture;
```

Zeigt auf das Gadget, für das aktuell der Empfang der Mausbotschaften aktiviert ist; ist für kein Gadget Empfang der Mausbotschaften festgelegt, ist *Capture* gleich 0.

Siehe auch: *TGadgetWindow::GadgetSetCapture*

Direction

```
TTileDirection GetDirection() const;
```

Die Orientierung der Gadgetanordnung – entweder horizontal oder vertikal.

DirtyLayout

```
bool DirtyLayout;
```

Zeigt an, daß sich das Layout geändert hat und die Gadgets neu angeordnet werden müssen. Die Verwendung von *DirtyLayout* vermeidet unnötiges Ausrichten, wenn Gadgetfenster generiert werden.

Siehe auch: *TGadgetWindow::LayoutSession*

Font

```
TFont* Font;
```

Zeigt auf die Schriftart, die zur Berechnung der Layout-Einheiten benutzt wird.

Siehe auch: *TGadgetWindow::GetFont*

FontHeight

```
uint GetFontHeight() const;
```

Enthält die Höhe der im Gadgetfenster benutzten Schrift.

Siehe auch: *TGadgetWindow::GetFont*

Gadgets

```
TGadget* Gadgets;
```

Zeigt auf das erste Gadget in der Gadgetliste.

HintMode

```
THintMode HintMode;
```

Enthält den Anzeigemodus für Hinweistexte.

Siehe auch: *THintMode* enum

Margins

```
TMargins Margins;
```

Enthält die Randwerte für das Gadgetfenster.

Siehe auch: *TGadgetWindow::SetMargins*

NumGadgets

```
uint NumGadgets;
```

Anzahl der Gadgets im Fenster.

ShrinkWrapHeight

```
bool ShrinkWrapHeight;
```

Wenn *ShrinkWrapHeight* **true** ist, paßt sich die Fensterbreite dem größten Gadget an, bei horizontal angeordneten Gadgets.

Siehe auch: *TGadgetWindow::SetShrinkWrap*

ShrinkWrapWidth

```
bool ShrinkWrapWidth;
```

Wenn *ShrinkWrapWidth* **true** ist, paßt sich die Fensterbreite dem breitesten Gadget an, bei vertikaler Anordnung der Gadgets.

Siehe auch: *TGadgetWindow::SetShrinkWrap*

WideAsPossible

```
uint WideAsPossible;
```

Die Anzahl der Gadgets größtmöglicher Breite.

Geschützte Elementfunktionen

Create

```
bool Create();
```

Überschreibt die *TWindow*-Elementfunktion und wählt die ursprüngliche Größe des Gadgets, wenn automatische Größenanpassung gewünscht war.

Siehe auch: *TGadgetWindow::SetShrinkWrap*

EvLButtonDown

```
void EvLButtonDown(uint modKeys, TPoint& point);
```

Reagiert auf die Mausbotschaft Drücken-der-linken-Maustaste, indem es das Ereignis an das unter dem Mauszeiger liegende Gadget weiterleitet.

EvLButtonUp

```
void EvLButtonUp(uint modKeys, TPoint& point);
```

Reagiert auf Mausbotschaft Loslassen-der-linken-Maustaste durch Weiterleitung des Mausereignisses an das Gadget, für das aktuell Empfang-der-Mausbotschaften aktiviert ist.

EvMouseMove

```
void EvMouseMove(uint modKeys, TPoint& point);
```

Wenn Mausbotschaften empfangen werden, reagiert *EvMouseMove* auf eine Mausbewegung durch Weiterleiten des Ereignisses an das Gadget, das aktuell den Mausempfang hat.

EvSize

```
void EvSize(uint sizeType, Tsize &);
```

Ruft *TWindow::EvSize* zum Ausführen des Standardvorgehens auf. Wenn *DirtyLayout* **true** und *WideAsPossible* größer 0 ist, setzt *EvSize DirtyLayout* auf **true** und ruft *TileGadgets* zur Neuanpassung der Größe auf sowie *Invalidate*, das den neu zu zeichnenden Bereich markiert.

Siehe auch: *TWindow::EvSize*

EvSysColorChange

```
void EvSysColorChange();
```

EvSysColorChange wird bei jeder Änderung der Systemfarben aufgerufen und gibt das Ereignis an alle Gadgets weiter.

GetDesiredSize

```
virtual void GetDesiredSize(TSize& size);
```

Wenn automatische Größenanpassung gewünscht war, liefert *GetDesiredSize* die Größe für die Anpassung der Ränder und Rahmen des in Breite und Höhe größten Gadget zurück; andernfalls liefert es die Breite und Höhe in der *Attr*-Struktur des Fensters zurück.

Wenn für eine bestimmte Gestaltung (beispielsweise eine Trennlinie zwischen den Gadgets, eine Schattenlinie, usw.) zusätzlicher Raum freigelassen werden soll, kann diese Funktion überschrieben werden. In diesem Fall muß aber wahrscheinlich auch *GetInnerRect* überschrieben werden, um die Berechnung des individuellen inneren Rechtecks zu ermöglichen.

Siehe auch: *TGadgetWindow::GetInnerRect*

GetInnerRect

```
virtual void GetInnerRect(TRect &rect);
```

GetInnerRect berechnet das von den Rändern und Rahmen des Gadgets eingeschlossene Rechteck.

Wenn für eine bestimmte Gestaltung (beispielsweise eine Trennlinie zwischen den Gadgets, eine Schattenlinie, usw.) zusätzlicher Raum freigelassen werden soll, kann diese Funktion überschrieben werden. In diesem Fall muß aber wahrscheinlich auch *GetDesiredSize* überschrieben werden, um die Berechnung der individuellen Gesamtgröße zu ermöglichen.

Siehe auch: *TGadgetWindow::GetDesiredSize*

GetMargins

```
void GetMargins(TMargins& margins, int& left, int& right, int& top, int&
bottom);
```

Liefert den linken, rechten, oberen und unteren Rand in Pixeln zurück.

LayoutUnitsToPixels

```
int LayoutUnitsToPixels(int units);
```

Rechnet Layout-Einheiten in Pixel um. Eine Layout-Einheit beträgt ein Achtel der im Fenster verwendeten Schrifthöhe.

Siehe auch: *TGadgetWindow::LayoutSession*

Paint

```
virtual void Paint(TDC& dc, bool erase, TRect& rect);
```

Setzt die Schriftart in den Gerätezusammenhang ein und ruft *Paint-Gadgets* auf.

Siehe auch: *TGadgetWindow::PaintGadgets*

PaintGadgets

```
virtual void PaintGadgets(TDC& dc, bool erase, TRect& rect);
```

Wenn es von *Paint* aufgerufen wird, geht *PaintGadgets* die Liste der Gadgets durch, berechnet ihre Fläche und färbt jedes Gadget neu ein.

Diese Funktion kann für eine individuelle Gestaltung (Trennlinie, Schattenlinie, usw.) überschrieben werden.

Siehe auch: *TGadgetWindow::EvPaint*

PositionGadget

```
virtual void PositionGadget(TGadget* previous, TGadget* next, TPoint&
point);
```

PositionGadget wird aufgerufen, um Abstandsanpassungen vor der Positionierung der einzelnen Gadgets vorzunehmen.

Siehe auch: *TGadgetWindow::TileGadgets*

TileGadgets

```
virtual void TileGadgets();
```

Richtet die Gadgets in der gewünschten Orientierung aus (horizontal oder vertikal). Ruft *PositionGadget* auf, um abgeleiteten Klassen die Abstandsanpassung der Gadgets in ihren Fenstern zu ermöglichen.

Siehe auch: *TGadgetWindow::PositionGadget*

Antworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_WM_LBUTTONDOWN	EvLButtonDown
EV_WM_LBUTTONUP	EvLButtonUp
EV_WM_MOUSEMOVE	EvMouseMove
EV_WM_SIZE	EvSize
EV_WM_SYSCOLORCHANGE	EvSysColorChange

Klasse TGauge gauge.h

TGauge, eine von *TControl* abgeleitete, streamfähige Klasse, legt das Verhalten von Meßanzeigen fest. Meßanzeigen dienen nur zur Ansicht, sind horizontal oder vertikal orientiert und geben analoge oder Zeit-Informationen über einen bestimmten Prozess. Im allgemeinen werden horizontale Meßanzeigen mit unterbrochenem (gestricheltem) Anzeigebalken zur Anzeige kurzer Vorgänge verwendet, während horizontale Meßanzeigen mit durchgehendem Anzeigebalken für die Darstellung langandauernder Vorgänge benutzt werden. Vertikale Meßanzeigen werden üblicherweise zur Darstellung analoger Prozesse eingesetzt.

Öffentliche Konstruktor

Konstruktor

```
TGauge(TWindow* parent, const char far* title, int id, int X, int Y,
       int W, int H, bool isHorizontal = true, int margin = 0, TModule*
module = 0);
```

Konstruiert ein *TGauge*-Oberflächenobjekt mit Rahmen, die durch den Wert von SM_CXBORDER festgelegt sind. Setzt *IsHorizontal* auf *isHorizontal*. Setzt die Rahmenstärke und den Abstand zwischen durchbrochenen Rahmen (LEDs) auf 0. Bestimmt den Bereich möglicher Werte zu 0 bis 100.

Öffentliche Elementfunktionen

GetRange

```
void GetRange(int& min, int& max) const;
```

Holt die Minimal- und Maximalwerte für die Meßanzeige.

GetValue

```
int GetValue() const
```

Holt den aktuellen Wert der Meßanzeige.

SetColor

```
void SetColor(TColor color);
```

Setzt das Datenelement *BarColor* auf den in *color* angegebenen Wert.

SetLed

```
void SetLed(int spacing, int thick = 90);
```

Setzt die *LedSpacing*- und *LedThick*-Datenelemente auf die Werte *spacing* und *thick*.

SetRange

```
void SetRange(int min, int max);
```

Setzt die *Min*- und *Max*-Datenelemente auf die *min*- und *max*-Werte, die vom Konstruktor zurückgeliefert werden. Wenn *Max* kleiner oder gleich *Min* ist, ersetzt *SetRange Max* durch *Min* plus 1.

SetValue

```
void SetValue(int value);
```

Beschränkt den Wert auf den Bereich zwischen den für die Meßanzeige festgelegten Höchst- und Niedrigstwerten. Wenn sich der aktuelle Wert geändert hat, markiert *SetValue* die alte Stelle für das Neuzeichnen. Danach setzt es das Datenelement *Value* auf den neuen Wert.

Geschützte Datenelemente

BarColor

```
TColor BarColor;
```

Enthält die Balken- oder LED-Farbe, die standardmäßig blau ist.

IsHorizontal

```
int IsHorizontal;
```

Ist auf das *ishorizontal*-Argument des Konstruktors gesetzt. *IsHorizontal* ist **true**, wenn die Meßanzeige horizontal ausgerichtet ist und **false**, wenn sie vertikal angeordnet ist.

LedSpacing

```
int LedSpacing;
```

Enthält den Integerwert des Abstands zwischen den durchbrochenen Anzeigebalken der Meßanzeige (in Meßanzeige-Einheiten).

LedThick

```
int LedThick;
```

Enthält die Dicke (Stärke) des durchbrochenen Anzeigebalkens.

Margin

```
int Margin;
```

Die Rahmenbreite und -höhe der Meßanzeige.

Max

```
int Max;
```

Enthält den auf der Meßanzeige angezeigten Höchstwert (in Meßanzeige-Einheiten).

Min

```
int Min;
```

Enthält den auf der Meßanzeige angezeigten Mindestwert (in Meßanzeige-Einheiten).

Value

```
int Value;
```

Enthält den aktuellen Wert der Meßanzeige.

Geschützte Elementfunktionen

EvEraseBkgnd

```
bool EvEraseBkgnd(HDC);
```

Umgeht die *EvEraseBkgnd*-Funktion von TWindow und löscht den Hintergrund der Meßanzeige. Immer wenn der Hintergrund neu aufgebaut wird, wird *EvEraseBkgnd* aufgerufen, um Flimmern zu vermeiden.

Paint

```
void Paint(TDC& dc, bool erase, TRect& rect);
```

Überschreibt die *Paint*-Funktion von TWindow und zeichnet Fläche und Rahmen der Meßanzeige. Zeichnet das angegebene Rechteck in den angegebenen Gerätekontext. Wertet *LedSpacing* und *IsHorizontal* aus, um eine horizontale oder vertikale Meßanzeige mit durchgehendem oder durchbrochenem Anzeigebalken zu zeichnen.

Siehe auch: *TGauge::LedSpacing, TGauge::IsHorizontal, TDC, TRect*

PaintBorder

```
virtual void Paint(TDC& dc);
```

Zeichnet den Rand der Meßanzeige unter Verwendung des angegebenen Gerätekontexts. *PaintBorder* zeichnet den Rahmen im angegebenen Stil (raised, embossed oder recessed). Wenn Sie einen Rahmenstil verwenden möchten, der von den ObjectWindows-Meßanzeigen nicht unterstützt wird, können Sie diese Funktion überschreiben.

Antworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_WM_ERASEBKGND	EvEraseBkgnd

Klasse TGdiObject gdiobjec.h

GdiObject ist die pseudo-abstrakte Stamm- und Basisklasse für die GDI-Rahmenobjekte von ObjectWindows (GDI = Graphics Device Interface, grafische Geräteschnittstelle). WIN-API-Aufrufe, denen ein GDI-Handle-Argument zugewiesen werden kann, werden typischerweise durch einfachere Aufrufe von ObjectWindows-Elementfunktionen ersetzt, in denen das Handle (und u.U. auch andere Argumente) vom aufrufenden Objekt zur Verfügung gestellt wird. Die *TGdiObject*-basierten Klassen ermöglichen das Arbeiten mit einem GDI-Handle und erzeugen ein C++ Objekt mit einem Alias-Handle. Einige GDI-Objekte basieren aus Gründen der Handle-Verwaltung ebenfalls auf *TGdiObject*. Die auf *TGdiObject* aufbauende Klassenhierarchie verwaltet im Allgemeinen alle GDI-Objekte, mit Ausnahme der DC(Device Context)-Objekte, die von der *TDC*-basierten Baumstruktur verwaltet werden.

Die fünf zur Wahl stehenden DC-Klassen *(TPen, TBrush, TFont, TPalette* und *TBitmap*) und die *TIcon-, TCursor-, TDib-* und *TRegion*-Klassen sind alle direkt aus *TGdiObject* abgeleitet.

TGdiObject pflegt das GDI-Handle sowie ein *ShouldDelete*-Flag, das festlegt, ob und wann Handle und Objekt gelöscht werden sollen. Den abge-

leiteten Klassen stehen geschützte Konstruktoren zur Verfügung: einer für eingelesene Handles sowie einer für allgemeine Verwendung.

Ein Mechanismus zur Kontrolle "verwaister" Objekte (orphan control) steht auf Wunsch zur Verfügung. Er ist standardmäßig aktiv, kann aber durch Definition des NO_GDI_ORPHAN_CONTROL-Bezeichners ausgeschaltet werden:

```
#define NO_GDI_ORPHAN_CONTROL
```

Bei eingeschalteter orphan control stehen die folgenden statischen Elementfunktionen zur Verfügung:

RefAdd, *RefCount*, *RefDec*, *RefFind*, *RefInc* und *RefRemove*.

Diese unterhalten Objekt-Referenzzähler und gestatten sicheres Wiederherstellen oder Löschen verwaister Objekte. Mit Makros, wie zum Beispiel OBJ_REF_ADD, kann durch einfaches Definieren oder Aufheben der Definition von NO_GDI_ORPHAN_CONTROL der Code zur Überwachung verwaister Objekte aktiviert, bzw. deaktiviert werden. Wenn NO_GDI_ORPHAN_CONTROL nicht definiert ist, erstreckt sich, beispielsweise, OBJ_REF_ADD(*handle*, *type*) auf *TGdiObject::RefAdd((handle),(type))*; NO_GDI_ORPHAN_CONTROL definiert ist, erstreckt sich das Makro auf *handle*.

Öffentlicher Destruktor

Destruktor

```
virtual ~TGdiObject();
```

Wenn *ShouldDelete* **false** ist, wird der Destruktor nicht aktiv. Ist *ShouldDelete* **true**, hängt die Arbeitsweise des Destruktors von der orphan control ab: Bei ausgeschalteter orphan control (also wenn NO_ORPHAN_CONTROL definiert ist) löscht *~TGdiObject* das GDI-Objekt; bei aktiver orphan control (Voreinstellung) wird das Objekt nur dann gelöscht, wenn der Referenzzähler 0 ist.

Typdefinitionen

enum TAutoDelete

```
enum TAutoDelete(NoAutoDelete, AutoDelete);
```

Dieser Aufzählungstyp ist in der privaten Basisklasse gdibadse.h definiert. Er zählt die Flag-Werte für GDI-Handle-Konstruktoren auf. Dieses Flag wird verwendet, um die Löschung der GDI-Objekte in den Destruktoren zu steuern.

enum TType

```
enum TType {None, Pen, Brush, Font, Palette, Bitmap, TextBrush};
```

Diese Aufzählung wird verwendet, um den Objekttyp in **struct** *TObjInfo* zu speichern. Mit dieser internen Struktur werden Objekt-Referenzzähler bei Debug-Sitzungen verfolgt.

Siehe auch: *TGdiObject::RefCount*

Öffentliche Elementfunktionen

GetObject

```
int GetObject(int count, void far* object) const;
```

Nimmt Informationen über dieses GDI-Objekt entgegen und plaziert sie im *object*-Puffer. Wenn der Aufruf erfolgreich ist und *object* ungleich 0 ist, liefert *GetObject* die in den Objektpuffer kopierte Byteanzahl zurück. Bei erfolgreichem Aufruf und *object* gleich 0, liefert *GetObject* die für den Typ des abgefragten Objekts im Objektpuffer benötigte Byteanzahl zurück. Je nach Typ des abgeleiteten GDI-Objekts liefert diese Funktion in *object* eine Struktur *LOGPEN*-, *LOGBRUSH*-, *LOGFONT*- oder *BITMAP*.

Siehe auch: *TPen::GetObject*, Struktur *BITMAP*, Struktur *LOGBRUSH*, Struktur *LOGFONT*, Struktur *LOGPEN*

RefAdd

```
static void RefAdd(HANDLE handle,TType type);
```

Steht nur bei eingeschalteter orphan control zur Verfügung (wenn also NO_GDI_ORPHAN_CONTROL nicht definiert ist). *RefAdd* fügt der *ObjInfoBag*-Tabelle einen Referenz-Eintrag für das Objekt mit dem angegebenen *handle* und *type* hinzu und setzt den Referenzzähler auf 1. Gibt es bereits einen passenden Eintrag in der Tabelle, wird die Funktion nicht aktiv.

Siehe auch: *TGdiObject::RefCount*, macro *OBJ_REF_ADD*

RefCount

```
static int RefCount(HANDLE handle);
```

Steht nur bei eingeschalteter orphan control zur Verfügung, also wenn NO_GDI_ORPHAN_CONTROL nicht definiert ist. *RefCount* gibt den aktuellen Referenzzähler dieses Objekts zurück oder -1, wenn das Objekt nicht in der *ObjInfoBag*-Tabelle steht.

Siehe auch: Makro *OBJ_REF_COUNT*

RefDec

```
static void RefDec(HANDLE handle);
static void RefDec (HANDLE handle, bool wantDelete);
```

Steht nur bei eingeschalteter orphan control zur Verfügung, also wenn NO_GDI_ORPHAN_CONTROL nicht definiert ist. *RefDec* vermindert den Referenzzähler des Objekts um 1 und löscht das Objekt, wenn der Referenzzähler 0 erreicht.Eine Warnung wird ausgegeben, wenn das Objekt hätte gelöscht werden sollen, aber nicht gelöscht wurde. Analog wird gewarnt, wenn das Objekt wider Erwarten gelöscht wurde. Das gelöschte Objekt wird aus der *ObjInfoBag*-Tabelle ausgetragen.

Die zweite Version von *RefDec* steht nur bei definiertem _ _TRACE-Bezeichner zur Verfügung. Die Standard-Löschroutine kann durch Setzen von *WantDelete* auf **true** oder **false** modifiziert werden.

Siehe auch: *TGdiObject::RefCount*, Makro *OBJ_REF_DEC*

RefFind

```
static TObjInfo* RefFind(HANDLE object);
```

Steht nur bei eingeschalteter orphan control zur Verfügung, also wenn NO_GDI_ORPHAN_CONTROL nicht definiert ist. Wird das Objekt gefunden, werden sein Typ und Referenzzähler in dem angegebenen *TObjInfo*-Objekt zurückgegeben. *RefFind* liefert 0 zurück, wenn kein Eintrag gefunden wird.

Siehe auch: *TGdiObject::RefCount*

RefInc

```
static void RefInc(HANDLE handle);
```

Steht nur bei eingeschalteter orphan control zur Verfügung, also wenn NO_GDI_ORPHAN_CONTROL nicht definiert ist. *RefInc* erhöht den zu *handle* gehörenden Referenzzähler um 1.

Siehe auch: *TGdiObject::RefCount*, Makro *OBJ_REF_INC*

RefRemove

```
static void RefRemove(HANDLE handle);
```

Steht nur bei eingeschalteter orphan control zur Verfügung, also wenn NO_GDI_ORPHAN_CONTROL nicht definiert ist. *RefRemove* entfernt den Referenzeintrag des Objektes mit dem gegebenen *handle* aus der *ObjInfoBag*-Tabelle. Wird das Handle nicht gefunden, bleibt die Funktion ohne Wirkung.

Siehe auch: *TGdiObject::RefCount*, Makro *OBJ_REF_REMOVE*

Geschützte Datenelemente

Handle

```
HANDLE Handle;
```

Das GDI-Handle dieses Objekts.

Siehe auch: *TGdiObject*, geschützte Konstruktoren

ShouldDelete

```
bool ShouldDelete;
```

Auf **true** gesetzt, wenn der Destruktor das GDI-Handle dieses Objektes entfernen muß.

Siehe auch: *TGdiObject*, geschützte Konstruktoren

Geschützte Elementfunktionen

CheckValid

```
void CheckValid( uint resId=IDS_FAILURE);
static void CheckValid( HANDLE handle, uint resId=IDS_GDIFAILURE);
```

Beide Versionen von *CheckValid* prüfen auf ein gültiges GDI-Objekt-Handle. Wird keines gefunden, wird für die angegebene Ressourcen-ID eine Exception ausgelöst.

Geschützte Konstruktoren

Konstruktor

`TGdiObject();`

Dieser Standard-Konstruktor setzt *Handle* auf 0 und *ShouldDelete* auf **true**. Er ist für die Verwendung durch abgeleitete Klassen vorgesehen, die das *Handle*-Element setzen müssen.

Konstruktor

`TGdiObject(HANDLE handle, TAutoDelete autoDelete = NoAutoDelete);`

Dieser Konstruktor ist ausschließlich für die Verwendung durch abgeleitete Klassen vorgesehen. Das *Handle*-Datenelement wird aus einem existierenden Handle eingelesen, das durch das Argument *handle* festgelegt ist. Das Datenelement *ShouldDelete* ist auf **false** voreingestellt. So ist sichergestellt, daß das eingelesene Handle durch Entfernen des Objekts nicht gelöscht wird.

Siehe auch: *TGDIObject::enumTAutoDelete, TGdiObject::Handle, TGdiObject::RefCount, TGdiObject::ShouldDelete*

Makros

OBJ_REF_ADD

`OBJ_REF_ADD(handle, type)`

Bei aktivierter orphan control (Voreinstellung), ist *OBJ_REF_ADD(handle, type)* als *TGdiObject::RefAdd((handle), (type))* definiert. Letzteres fügt der *ObjInfoBag*-Tabelle einen Referenzeintrag für das Objekt mit dem gegebenen *handle* und *type* hinzu und setzt seinen Zähler auf 1.

Bei ausgeschalteter orphan control ist *OBJ_REF_ADD(handle)* als *handle* definiert. Mit diesem Makro läßt sich orphan control-Code schreiben, der mit dem einzelnen Statement `#define NO_GDI_ORPHAN_CONTROL` auf einfache Weise deaktiviert werden kann.

Siehe auch: *TGdiObject::RefAdd*

OBJ_REF_COUNT

`OBJ_REF_COUNT(handle)`

Bei aktiver orphan control (Voreinstellung), ist *OBJ_REF_COUNT(handle)* als *TGdiObject::RefCount((handle))* definiert. Letzteres liefert den Referenzzähler des Objekts mit dem gegebenen *handle* zurück oder -1, wenn kein solches Objekt existiert. Bei deaktivierter orphan control ist *OBJ_REF_COUNT* T-Handle als -1 definiert. Mit diesem Makro läßt sich orphan control-Code schreiben, der mit dem einzelnen Statement `#define NO_GDI_ORPHAN_CONTROL` auf einfache Weise deaktiviert werden kann.

Siehe auch: *TGdiObject::RefCount*

OBJ_REF_DEC

`OBJ_REF_DEC(handle, wantDelete)`

Bei eingeschalteter orphan control (der Voreinstellung) ist *OBJ_REF_DEC(handle, wantDelete)* entweder definiert als *TGdiObject::RefDec((handle))* oder als *TGdiObject::RefDec((handle), (wantDelete))*. Das zweite Format kommt nur bei definiertem _ _TRACE vor. *RefDec(handle)* vermindert den Referenzzähler des zu *handle* gehörenden Objektes und löscht auf Wunsch verwaiste Objekte oder informiert über ihre Existenz. Bei deaktivierter orphan control ist *OBJ_REF_DEC(handle)* als *handle* definiert. Mit diesem Makro läßt sich orphan control-Code schreiben, der mit dem einzelnen Statement `#define NO_GDI_ORPHAN_CONTROL` auf einfache Weise deaktiviert werden kann.

Siehe auch: *TGdiObject::RefDec*

OBJ_REF_INC

`OBJ_REF_INC(handle)`

Bei eingeschalteter orphan control (voreingestellt), ist *OBJ_REF_INC(handle)* als *TGdiObject::RefInc((handle))* definiert.Letzteres inkrementiert den Referenzzähler des zu *handle* gehörenden Objekts. Ist orphan control deaktiviert, ist *OBJ_REF_DEC(handle)* als *handle* definiert. Mit diesem Makro läßt sich orphan control-Code schreiben, der mit dem einzelnen Statement `#define NO_GDI_ORPHAN_CONTROL` auf einfache Weise deaktiviert werden kann.

Siehe auch: *TGdiObject::RefInc*

OBJ_REF_REMOVE

`OBJ_REF_REMOVE(handle)`

Wenn die orphan control eingeschaltet (voreingestellt) ist, ist *OBJ_REF_REMOVE(handle)* als *TGdiObj::RefRemove((handle))* definiert. Letzteres entfernt den Referenzzähler für das zu *handle* gehörende Objekt aus der *ObjInfoBag*-Tabelle. Bei ausgeschalteter orphan control ist *OBJ_REF_REMOVE(handle)* als *handle* definiert. Mit diesem Makro läßt sich orphan control-Code schreiben, der mit dem einzelnen Statement `#define NO_GDI_ORPHAN_CONTROL` auf einfache Weise deaktiviert werden kann.

Siehe auch: *TGdiObj::RefRemove*

Klasse TGdiObject::TXGdi gdibase.h

Beschreibt eine Exception, die aus GDI-Fehlern resultiert, wie beispielsweise der Erzeugung einer zu großen Zahl von *TWindow* DCs. Diese Exception tritt beispielsweise auf, wenn ein DC-Treiber nicht lokalisiert oder eine DIB-Datei nicht gelesen werden kann.

Das folgende Code-Segment aus dem PAINT.CPP-Beispielprogramm auf der mitgelieferten Diskette löst eine TXGdi-Exception aus, wenn kein neues DIB erzeugt werden kann.

```
void TCanvas::NewDib(int width, int height, int nColors)
{
   TDib* dib;
   try {
     dib = new TDib(width, height, nColors);
   }
   catch (TGdiObject::TXGdi& x) {
     Messagebox("Could Not Create DIB", GetApplication()->Name, MB_OK);
     return;
   }
```

Öffentliche Konstruktoren

Konstruktor

```
TXGdi(uint resId = IDS_GDIFAILURE, HANDLE = 0);
```

Erzeugt ein *TXGdi*-Objekt mit der *IDS_GDIFAILURE*-Standardmeldung.

Öffentliche Elementfunktionen

Clone

```
TXOwl* Clone;
```

Kopiert ein Exception-Objekt. *Clone* muß in jeder Ableitungsklasse von *TXOwl* implementiert sein.

Msg

```
static string Msg(uint resId, HANDLE);
```

Konvertiert die Ressourcen-ID in einen String und liefert die Stringmeldung zurück.

Throw

```
void Throw();
```

Setzt das Exception-Object ab. *Throw* muß in jeder aus *TXOwl* abgeleiteten Klasse implementiert sein.

Klasse TGroupBox — groupbox.h

Eine Instanz einer *TGroupBox* ist ein Oberflächenobjekt, das ein entsprechendes Gruppenelement in Windows repräsentiert. Im allgemeinen werden *TGroupBox*-Objekte nicht in Dialogfeldern oder Dialogfenstern *(TDialog)* verwendet, sondern für die Erzeugung von Gruppen in einem Fenster.

Zwar dienen Gruppenfenster keinem unmittelbaren Zweck auf dem Bildschirm, sie fassen eine Gruppe von Auswahlfeldern - wie Schalt- und Markierungsfelder - optisch zusammen. Unter der Oberfläche können sie aber eine wichtige Rolle in der Behandlung von Zustandsänderungen für ihre Gruppe von Dialogelementen (üblicherweise Schalt- und Markierungsfelder) übernehmen.

Es sollen zum Beispiel alle Schaltfelder in einer Gruppe in ähnlicher Weise auf eine Auswahländerung reagieren. Das kann durch Ableitung einer Klasse aus *TGroupBox* erreicht werden, die die Elementfunktion *SelectionChanged* überschreibt.

Alternativ könnte auf eine Auswahländerung in einer Gruppe von Schaltfeldern auch durch Definition des Antwortverhaltens des übergeordneten Fensters reagiert werden. Zu diesem Zweck wäre eine auf der Kennung des untergeordneten Fensters aufbauende Elementfunktion zu definieren, die die Kennung der Gruppe verwendet. Die Gruppe sendet dann automatisch bei jedem Wechsel einer Schaltflächenauswahl eine auf ihrer Kennung basierende Meldung an ihr übergeordnetes Fenster. Diese Klasse ist streamfähig.

Öffentliche Datenelemente

NotifyParent

```
bool NotifyParent;
```

Ein Flag, das anzeigt, ob das übergeordnete Fenster bei einer Zustandsänderung der Auswahlfelder der Gruppe benachrichtigt wird. Standardmäßig ist *NotifyParent* **true**.

Öffentliche Konstruktoren

Konstruktor

```
TGroupBox(TWindow* parent, int Id, const char far *text, int x, int y,
          int w, int h, TModule* module = 0);
```

Erzeugt ein Gruppenfensterobjekt mit den Angaben für das übergeordnete Fenster *(Parent)*, die Kontrollkennung *(Id)*, den zugewiesenen Text *(text)*, die Position *(x,y)* relativ zum Ursprung des Client-Bereichs des übergeordneten Fensters, die Breite *(w)* und die Höhe *(h)*. Ruft den *TControl*-Konstruktor mit ähnlichen Parametern auf und ändert anschließend *Attr.Style* durch Hinzufügen von BS_GROUPBOX und Entfernen von WS_TABSTOP. *NotifyParent* ist auf **true** gesetzt; standardmäßig wird das übergeordnete Fenster benachrichtigt, wenn eine Änderung der Auswahl in einem der Steuerelemente eines Gruppenfensters stattfindet.

Konstruktor

```
TGroupBox(TWindow* parent, int resourceId, TModule* module = 0);
```

Konstruiert ein *TGroupBox*-Objekt zur Zuweisung an ein Steuerelement einer Gruppe aus einem *TDialog*. Ruft den *TControl*-Konstruktor mit identischen Parametern auf. *resourceID* muß einer benutzerdefinierten Gruppenressource entsprechen.

Siehe auch: *TControl::TControl, TWindow::DisableTransfer*

Öffentliche Elementfunktionen

GetClassName

```
char far* GetClassName();
```

GetClassName gibt den Namen der Windows-Registrierklasse von *TGroupBox* zurück, "BUTTON". Wenn BWCC eingeschaltet ist, liefert *GetClassName* BUTTON_CLASS zurück.

SelectionChanged

```
virtual void SelectionChanged(int controlId);
```

Wenn *NotifyParent* **true** ist, informiert *SelectionChanged* durch Versenden einer auf der Gruppenkennung aufbauenden Nachricht das übergeordnete Fenster einer Gruppe, daß sich eine Auswahl geändert hat . Diese Elementfunktion kann überschrieben werden, damit die Gruppe Auswahländerungen ihrer Steuerelemente bearbeiten kann.

Klasse THatch8x8Brush gdiobjec.h

THatch8x8Brush ist von *TBrush* abgeleitet und definiert einen kleinen, monochromen, konfigurierbaren Schraffurpinsel (einen Pinsel, der einen Bereich mit einem aus Schraffurzeichen bestehenden Muster füllt) mit einem 8x8-Raster. Der Schraffurpisel ist ein »logischer« Pinsel, der aus geräteunabhängigen Bitmaps (DIBs) erzeugt wird. Er kann daher in jeden beliebigen Gerätekontext übergeben werden, der dem Pinsel dann die für das jeweilige Gerät geeignete Form gibt.

Der Standardpinsel hat einen weißen Vorder- und einen schwarzen Hintergrund, die Farben des Schraffurpinsels sind aber einstellbar. Alle in *TColor*-Objekten gekapselten Farben - also alle Standard-RGB-Werte - können verwendet werden.

THatch8x8Brush enthält statische Arrays, die gängige Schraffuren definieren. Zu den zur Wahl stehenden Schraffuren gehören

Vorwärts diagonal `/ / / / / /`

`// // // // //`

Rückwärts diagonal `\ \ \ \ \ \`

`\\ \\ \\ \\ \\`

THatch8x8Brush kann beispielsweise verwendet werden, um eine Vielzahl schraffierter Pinselmuster als Rahmen für einfache Rechtecke oder OLE-Container zu entwerfen. *THatch8x8Brush* kann auch in Verbindung mit *TUIHandle* eingesetzt werden.

Öffentliche Datenelemente

Hatch11F1[8]

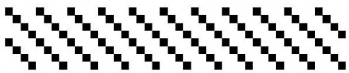
```
const static uint8 Hatch11F1[8];
```

Das statische Array *Hatch11F1[8]* enthält ein logisches Schraffurmuster, bei dem immer ein gesetztes mit einem nicht-gesetzten Bit abwechselt. Die Reihen sind um jeweils 1 Pixel gegeneinander versetzt, wie im folgenden Beispiel abgebildet.

Hatch13B1[8]

```
const static uint8 Hatch13B1[8];
```

Das statische Array *Hatch13B1[8]* enthält ein logisches Schraffurmuster, bei dem immer ein gesetztes mit drei nicht-gesetzten Bits abwechselt. Die Reihen sind um jeweils 1 Pixel nach rechts gegeneinander versetzt. Es ergibt sich eine rückwärts gerichtete Schraffur, wie im folgenden Beispiel abgebildet.



Hatch13F1[8]

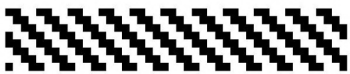
```
const static uint8 Hatch13F1[8];
```

Das statische Array *Hatch13F1[8]* enthält ein logisches Schraffurmuster, bei dem immer ein gesetztes mit drei nicht-gesetzten Bits abwechselt. Die Reihen sind um jeweils 1 Pixel nach links gegeneinander versetzt. Es ergibt sich eine vorwärts gerichtete Schraffur, wie im folgenden Beispiel abgebildet.

Hatch22B1[8]

```
const static uint8 Hatch22B1[8];
```

Das statische Array *Hatch22B1[8]* enthält ein logisches Schraffurmuster, bei dem immer zwei gesetzte mit zwei nicht-gesetzten Bits abwechseln. Die Reihen sind um jeweils 1 Pixel nach rechts gegeneinander versetzt. Es ergibt sich eine rückwärts gerichtete Schraffur, wie im folgenden Beispiel abgebildet.



Hatch22F1[8]

```
const static uint8 Hatch22F1[8];
```

Das statische Array *Hatch22F1[8]* enthält ein logisches Schraffurmuster, bei dem immer zwei gesetzte mit zwei nicht-gesetzten Bits abwechseln. Die Reihen sind um jeweils 1 Pixel nach links gegeneinander versetzt. Es

ergibt sich eine vorwärts gerichtete Schraffur, wie im folgenden Beispiel abgebildet.

Öffentlicher Konstruktor

Konstruktor

```
THatch8x8Brush(const uint8 hatch[], TColor fgColor=TColor::White, TColor
bgColor=TColor::Black);
```

Konstruiert ein *THatchBrush*-Objekt mit den angegebenen Farben und dem angegebenen Schraffurmuster. Der Schraffurpinsel ist standardmäßig weiß auf schwarzem Untergrund, die angezeigten Farben können aber durch Übergeben anderer Farbwerte in den Argumenten des Konstruktors gesteuert werden. *fgColor* steht für die Vordergrundfarbe, *bgColor* für den Untergrund eines Objekts vom Typ *TColor*. Farben können entweder im Palettenmodus (als Referenz auf einen entsprechenden Farbpaletteneintrag in der aktuell realisierten Palette) oder im RGB-Modus (als tatsächlicher Rot-, Grün- oder Blauwert) angegeben werden.

Öffentliche Elementfunktion

Reconstruct

```
void Reconstruct(const uint8 hatch[], TColor fgColor, TColor bgColor);
```

Baut den Schraffurpinsel mit neuem Muster oder neuen Farben wieder auf.

Siehe auch: *TBrush, TColor, TUIHandle*

Klasse TIC — dc.h

TIC ist eine von *TDC* abgeleitete DC-Klasse (DC= Device Context, Gerätekontext), die einen Konstruktor zur Erzeugung eines DC-Objekts aus expliziten Treiber-, Geräte- und Portnamen zur Verfügung stellt.

Öffentliche Konstruktoren

Konstruktor

```
TIC(const char far* driver, const char far* device, const char far*
    output, const DEVMODE far* initData=0);
```

Erzeugt ein DC-Objekt mit den angegebenen Namen und Initialisierungswerten der Treiber, Geräte und Ports.

Siehe auch: *TDC::GetDeviceCaps*, Struktur *DEVMODE*

Klasse TIcon gdiobjec.h

TIcon, aus *TGdiObject* abgeleitet, stellt die GDI-Objektsymbol-Klasse dar. *TIcon*-Konstruktoren können Symbole aus einer Ressource oder aus expliziter Information generieren. Der *TIcon*-Destruktor überlädt den Basisdestruktor *~TGdiObject()*, da Symbole keine realen GDI-Objekte sind.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TIcon(HICON handle, TAutoDelete autoDelete = NoAutoDelete);
```

Erzeugt ein *TIcon*-Objekt und setzt das *Handle*-Datenelement auf das gegebene, eingelesene *handle*. Das *ShouldDelete*-Datenelement ist auf **false** voreingestellt, wodurch sichergestellt wird, daß das eingelesene Handle nicht gelöscht wird, wenn das C++ Object entfernt wird.

Konstruktor

```
TIcon(HINSTANCE instance, const TIcon& icon);
```

Erzeugt eine Kopie des angegebenen *icon*-Objekts.

Konstruktor

```
TIcon(HINSTANCE instance, TResID resID);
```

Erzeugt ein Symbol-Objekt aus der gegebenen Ressource.

Konstruktor

```
TIcon(HINSTANCE instance, const char far* filename, int index);
```

Erzeugt ein Symbol-Objekt aus der angegebenen Ressourcendatei.

Konstruktor

```
TIcon (HINSTANCE instance, const TSize& size, int planes, int bitsPixel,
const void far* andBits, const void far* xorBits);
```

Erzeugt ein Symbol-Objekt mit den gegebenen Werten.

Konstruktor

```
TIcon(const void* resBits, DWORD resSize);
```

Erzeugt ein Symbolobjekt in der vorgegebenen Größe aus den aus dem resBits-Puffer ausgelesenen Bits.

Konstruktor

```
TIcon(const ICONINFO* iconInfo);
```

Erzeugt ein Objekt aus der angegebenen ICONINFO-Information.

Destruktor

```
~TIcon();
```

Entfernt das Symbol und setzt allen Speicher frei, den das Symbol belegt hat.

Siehe auch: *~TGdiObject, TGdiObject::Handle, TGdiObject::ShouldDelete, TResId, TSize, Struktur ICONINFO*

Öffentliche Elementfunktionen

GetIconInfo

```
bool GetIconInfo(ICONINFO* iconInfo) const;
```

Ermittelt Informationen über dieses Symbol und kopiert sie in die vorgegebene ICONINFO-Struktur. Gibt **true** zurück, wenn der Aufruf erfolgreich war, andernfalls **false**.

Siehe auch: Struktur *ICONINFO*

Operator HICON()

```
operator HICON() const;
```

Typumwandlungs-Operator, der das *Handle* dieses Symbols in den Typ *HICON* (der Windows-Datentyp, der das Handle für eine Symbolressource darstellt) umwandelt.

Klasse TInputDialog inputdia.h

TInputDialog stellt ein generisches Dialogfenster zur Verfügung, um Texteingaben des Anwenders zu ermitteln. Bei der Generierung des Texteingabefensters werden sein Titel, seine Eingabeaufforderung und sein Standard-Eingabetext spezifiziert. *TInputDialog* ist eine streamfähige Klasse.

Öffentliche Datenelemente

buffer

```
char far* buffer;
```

Zeiger auf den Puffer, der den vom Anwender eingegebenen Text zurückliefert. An den Konstruktor des Texteingabefensters übergeben, enthält er den Standardtext, der beim Öffnen des Editierfensters angezeigt wird.

BufferSize

```
int BufferSize;
```

Enthält die Größe des Puffers, der die Eingaben des Anwenders zurückliefert.

prompt

```
char far* prompt;
```

Zeigt auf die Eingabeaufforderung des Texteingabefensters.

Öffentliche Konstruktoren

Konstruktor

```
TInputDialog(TWindow* parent, const char far *title,
             const char far *prompt, char far* buffer, int bufferSize,
             TModule* module = 0, TValidator* valid = 0);
```

Ruft den Konstruktor von *TDialog* auf und übergibt ihm *parent*, den Ressourcen-Bezeichner und *module*. Setzt den Titel des Dialogfensters auf *title* und das statische Element der Eingabeaufforderung auf *prompt*. Setzt die *Buffer*- und *BufferSize*-Datenelemente auf *buffer* und *bufferSize*.

Siehe auch: *TDialog::TDialog*

Öffentliche Elementfunktion

TransferData

```
void TransferData(TTransferDirection direction);
```

Überträgt die Daten des Texteingabefensters. Wenn *direction tdSetData* ist, wird der Text der statischen Elemente und des Editierfensters auf den Text in *prompt* und *buffer* gesetzt. Wenn *direction* tdGetData ist, wird *buffer* mit dem aktuellen Text von *Editor* gefüllt.

Geschützte Elementfunktion

SetupWindow

```
virtual void SetupWindow();
```

Beim Aufbau des Fensters ruft *SetupWindow TDialog::SetupWindow* auf und beschränkt dann die Anzahl der Zeichen, die vom Benutzer eingegeben werden können, auf *bufferSize* -1.

Klasse TInStream — docview.h

TInStream ist eine aus *TStream* und *istream* abgeleitete Basisklasse, die zur Definition von Eingabe-Streams für Dokumente verwendet wird.

Öffentliche Konstruktoren

Konstruktor

```
TInStream(TDocument& doc, const char far* name, int mode);
```

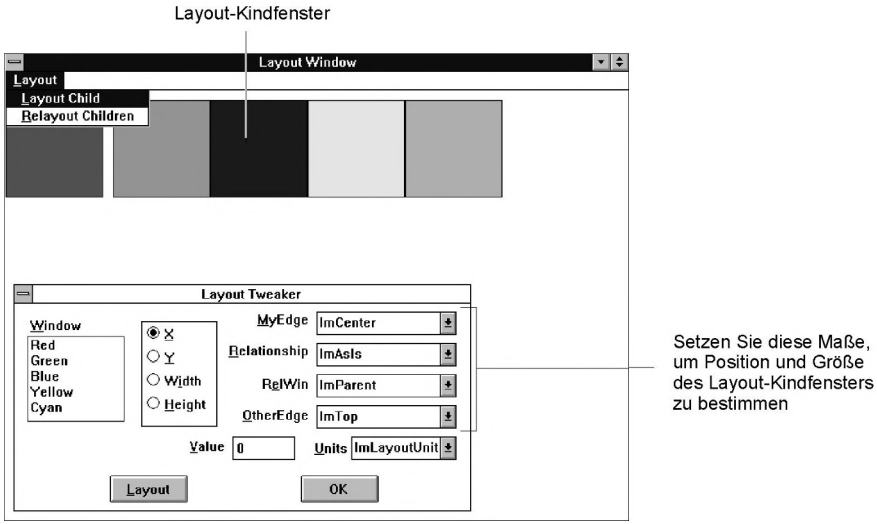
Erzeugt ein *TInStream*-Objekt. *doc* bezieht sich auf das Dokumentobjekt, *name* ist der benutzerdefinierte Name des Streams und *mode* ist der Öffnungsmodus für den Stream.

Siehe auch: *TOutStream*, *ofXXXX document open* enum, *shdocument sharing* enum

Struktur TLayoutConstraint — layoutco.h

TLayoutConstraint ist eine Struktur, die Randbedingungen für das Layout definiert. Layout-Randbedingungen werden als Verhältnis von Kantenlänge oder Größe eines Fensters zur Kantenlänge oder Größe eines gleichberechtigten oder übergeordneten Fensters angegeben. Wenn eine Abhängigkeit von Fenstern untereinander festgelegt wird, hängen die Abmessungen des untergeordneten Fensters vom übergeordneten Fenster ab. Die Größe in einer Dimension kann durch die andere Dimension bestimmt sein: zum Beispiel kann die Breite eines Fensters zweimal seine Höhe betragen. *TLayoutMetrics* verzeichnet die möglichen Verhältnisse zwischen Größen- und Kantenbedingungen.

LAYOUT.CPP im Verzeichnis OWLAPI\LAYOUT zeigt Ihnen im folgenden Beispiel, wie Layoutbedingungen eingerichtet werden.



Öffentliche Datenelemente

MyEdge

```
uint MyEdge;
```

MyEdge enthält den Namen der Randbedingung *(lmTop, lmBottom, lmLeft, lmRight, lmCenter, lmWidth oder lmHeight)* für die Größe oder die Kantenlänge eines Fensters.

Siehe auch: *TWidthHeight* enum

OtherEdge

```
uint OtherEdge;
```

OtherEdge enthält den Namen der Randbedingung *(lmTop, lmBottom, lmLeft, lmRight, lmCenter, lmWidth oder lmHeight)* für die Größe oder die Kantenlänge des anderen Fensters.

Siehe auch: *TWidthHeight* enum

Relationship

```
TRelationship Relationship;
```

Relationship gibt die Art der Abhängigkeit zwischen zwei Fenstern an *(lmRightOf, lmLeftOf, lmAbove, lmBelow, lmSameAs oder lmPercentOf)*. Der Wert *lmAbsolute* zeigt an, daß keine Abhängigkeit existiert.

Siehe auch: *TRelationship* enum

RelWin

```
TWindow *RelWin;
```

RelWin ist ein Zeiger auf die gleichgeordneten Fenster oder auf *lmParent*, wenn das abhängige Fenster durch die Abmessungen des übergeordneten Fensters definiert ist. *RelWin* zeigt auf das Fenster selbst (**this**), wenn die Abmessungen eines untergeordneten Fensters in einer Dimension von

einer seiner anderen Dimensionen abhängen (zum Beispiel seine Höhe durch seine Breite bestimmt ist).

Siehe auch: *TRelationship* enum

Units

```
TMeasurementUnits Units;
```

Units zählt die Maßeinheiten (entweder Pixel oder Layout-Einheiten) auf, die zur Höhen- und Breitenbestimmung der Fenster benutzt werden. Im Gegensatz zu Pixeln basieren Layouteinheiten auf der Größe der Systemschrift.

Ihre Größe wird von einem Wechsel der Bildschirmauflösung nicht beeinflußt.

Siehe auch: *TMeasurementUnits* enum

union

```
union {
        int  Margin;
        int  Value;
        int  Percent;
      };
```

Diese Variante wurde eingeführt, um eine Namensvergabe für die Layout-Randbedingungen zu ermöglichen. Margin wird für die Aufzählungswerte *lmAbove*, *lmLeftOf* oder *lmRightOf* in *TRelationship* verwendet. Value wird für die Aufzählungswerte *lmSameAs* oder *lmAbsolute* in *TRelationship* benutzt, Percent für *lmPercentOf*.

Siehe auch: *TMeasurementUnits* enum

Klasse TLayoutMetrics — layoutwi.h

TLayoutMetrics enthält die vier Layout-Randbedingungen, die zur Festlegung der Layoutmaße eines Fensters benutzt werden. Diese Tabelle verzeichnet die Randbedingungen, die für die *X*-, *Y*-, *Height*- und *Width*-Felder benutzt werden können.

Feld	Randbedingungen
X	lmLeft, lmCenter, lmRight
Y	lmTop, lmCenter, lmBottom
Höhe	lmCenter, lmRight, lmWidth
Breite	lmCenter, lmBottom, lmHeight

Hängen die Abmessungen eines Fensters von einem übergeordneten Fenster ab, muß der Fensterverhältnis-Zeiger (*lmParent*) *lmParent* sein (nicht der aktuelle Zeiger des übergeordneten Fensters). Beispiel:

```
TWindow* child = new
TWindow(this, "");
TLayoutMetrics metrics;
metrics.X.Set(lmCenter, lmSameAs, lmParent, lmCenter);
metrics.Y.Set(lmCenter, lmSameAs, lmParent, lmCenter);
SetChildLayoutMetrics(*child, metrics);
```

Der Zeiger des übergeordneten Fensters (**this**) sollte nicht als Verhältniszeiger des abhängigen Fensters benutzt werden.

Öffentliche Datenelemente

Height

```
TEdgeOrWidthConstraint Height;
```

Enthält die Randbedingung für Höhe, Abstand Mitte-Kante (center edge) oder Basiskante des Fensters.

Width

```
TEdgeOrWidthConstraint Width;
```

Enthält die Randbedingung für Breite, Abstand Mitte-Kante (center edge) oder rechte Kante (*lmRight*) des Fensters.

X, Y

```
TEdgeConstraint X, Y;
```

X enthält die Randbedingung für die X-Kanten (links, Mitte, rechts) des Fensters. *Y* enthält die Randbedingung für die Y-Kanten (oben, Mitte, unten) des Fensters.

Öffentliche Konstruktoren

Konstruktor

```
TLayoutMetrics();
```

Erzeugt und initialisiert ein *TLayoutMetrics*-Objekt. Das geschieht, indem die Maßangaben für das übergeordnete und das abhängige Fenster auf die angegebenen Layouteinheiten gesetzt werden und das Verhältnis zwischen beiden Fenstern auf die in *lmAsIs* (von *TRelationship*) festgelegten Werte. Stellt die folgenden Standardwerte ein:

```
X.RelWin = 0;
X.MyEdge = lmLeft;
X.Relationship = lmAsIs;
X.Units = lmLayoutUnits;
X.Value = 0;
Y.RelWin = 0;
Y.MyEdge = lmTop;
Y.Relationship = lmAsIs;
Y.Units = lmLayoutUnits;
Y.Value = 0;
Width.RelWin = 0;
Width.MyEdge = lmWidth;
Width.Relationship = lmAsIs;
```

```
Width.Units = lmLayoutUnits;
Width.Value = 0;
Height.RelWin = 0;
Height.MyEdge = lmHeight;
Height.Relationship = lmAsIs;
Height.Units = lmLayoutUnits
Height.Value = 0;
```

Das folgende Progamm generiert zwei abhängige Fenster und einen Rahmen, in dem weitere Layout-Randbedingungen angegeben werden können.

```
#include <owl\owl.h>
#include <owl\framewin.h>
#include <owl\applicat.h>
#include <owl\layoutwi.h>
#include <owl\decorate.h>
#include <owl\decmdifr.h>
#include <owl\layoutco.h>
#pragma hdrstop

//Erzeugen einer Ableitungsklasse//

class TMyDecoratedFrame : public TDecoratedFrame {
public:
   TMyDecoratedFrame(TWindow* parent, const char far* titel,
                     TWindow& clientWnd, TWindow* MyChildWindow);
   void SetupWindow();
   {
      TDecoratedFrame::SetupWindow();
      MyChildWindow-ShowWindow(SW_NORMAL);
      MyChildWindow-BringWindowToTop();
   }
};

// Aufbau eines Rahmenfensters //

  TMyDecoratedFrame::TmyDecoratedFrame(TWindow * parent, const char far
* title,
TWindow& clientWnd)
    : TDecoratedFrame(parent, titel, clientWnd),
      TFrameWindow(parent, titel, &clientWnD),
      TWindow(parent, title)
{
// Erzeugen eines neuen TMyChildWindow-Fensters//

    MyChildWindow = new TWindow(this, "");
    MyChildWindow-Attr.Style |= WS_BORDER |WS_VISIBLE |WS_CHILD;
    MyChildWindow-SetBkgndColor(RGB(0,100,0));

// Maße des abhängigen Fensters festlegen //

    TLayoutMetrics layoutMetrics;
    layoutMetrics.X.Absolute(lmLeft, 10);
    layoutMetrics.Y.Absolute(lmTop, 10);
```

```
        layoutMetrics.Width.Absolute( 80 );
        layoutMetrics.Height.Absolute( 80 );
    }
        SetChildLayoutMetrics(*MyChildWindow, layoutMetrics);

    class TMyApp : public TApplication {
    public:

        virtual void InitMainWindow()
        {
          TWindow* client = new TWindow(0, "title");
          MainWindow = new TMyDecoratedFrame(0, "Layout Window Ex",
*client);
        }
    };
    int OwlMain(int, char**) {

       return TMyApp.Run();
    }
```

Klasse TLayoutWindow — layoutwi.h

TLayoutWindow ist aus *TWindow* abgeleitet und stellt Optionen zur Definition der Layout-Maße eines Fensters zur Verfügung. Vergleiche *TLayoutConstraint* zur Definiton der Layout-Randbedingungen und *TLayoutMetrics* zur Beschreibung der Maße, die zur Bestimmung von Layout-Randbedingungen verwendet werden können.

Die folgenden Beispiele zeigen, wie verschiedene Maße unter Verwendung von Kantenbedingungen festgelegt werden. Die Beispiele gehen von einem abhängigen und einem übergeordneten Fenster aus, aber ebenso ist ein gleichberechtigtes Verhältnis der beiden Fenster (zwei untergeordnete Fenster) möglich. Es ist zu beachten, daß das abhängige Fenster sich mitbewegt, wenn der Ursprung des übergeordneten Fensters verändert wird (linke Kante und Oberkante).

Beispiele

Beispiel 1

Zur Erzeugung eines größenveränderlichen Fensters sind die linke und die obere Begrenzung des abhängigen Fensters in ein festes Verhältnis zur linken Kante und zur Oberkante des übergeordneten Fensters zu setzen. Werden in diesem Beispiel die rechte und die Unterkante des übergeordneten Fensters verlängert, wachsen die rechte und die Unterkante des abhängigen Fensters um den gleichen Betrag. Sowohl die X- als auch die Y-Bedingung beträgt 10 Layout-Einheiten der Kanten des übergeordneten Fensters, die Bedingungen für Breite und Höhe sind auf 40 Einheiten festgelegt. Insbesondere bestimmt sich die Breite (*lmWidth*) als Abstand von

40 Einheiten von der rechten Kante des übergeordneten Fensters (*lmLeftOf* = *lmSameAs* + offset oder *sameas* - 40).

Die folgenden Layout-Bedingungen werden verwendet:

```
layoutmetrics.X.Set(lmLeft, lmRightOf, lmParent, lmLeft, 10);
layoutmetrics.Y.Set(lmTop, lmBelow, lmParent, lmTop, 10);
layoutmetrics.Width.Set(lmRight, lmLeftOf, lmParent, lmRight, 40);
layoutmetrics.Height.Set(lmBottom, lmAbove, lmParent, lmBottom, 40);
SetChildLayoutMetrics(*MyChildWindow, layoutMetrics);
```

Beispiel 2

Zur Erzeugung von Fenstern fester Größe und Lage wird die rechte Kante des abhängigen Fensters auf einen festen Abstand zur linken Kante des übergeordneten Fensters und seine Unterkante auf einen festen Abstand zu dessen Oberkante gesetzt. In diesem Beispiel sind die Bedingungen für die X- und die Y-Kante auf 10, die für Breite und Höhe auf 100 festgelegt.

Die folgenden Layout-Bedingungen werden verwendet:

```
layoutmetrics.X.Set(lmLeft, lmRightOf, lmParent, lmLeft, 10);
layoutmetrics.Y.Set(lm Top, lmBelow, lmParent, lmTop, 10);
layoutmetrics.Width.Absolute(100);
layoutmetrics.Height.Absolute(100);
SetChildLayoutMetrics(*MyChildWindow, layoutMetrics);
```

Beispiel 3

Um ein Fenster fester Größe und mit gleichbleibendem Abstand zur rechten Ecke des übergeordneten Fensters zu generieren, werden die Ober- und Unterkante des abhängigen Fensters auf festen Abstand (*lmLayout*-Einheiten oder Pixel) zur Unterkante des übergeordneten Fensters sowie seine linke und rechte Kante auf festen Abstand zu dessen rechter Kante gesetzt. In diesem Beispiel ist in die Bedingungen für die Breite und die Höhe 100 und in die X- und die Y-Bedingung jeweils 10 eingesetzt. So bewegt sich das abhängige Fenster, dessen Größe nicht veränderlich ist, zusammen mit der unteren rechten Ecke des übergeordneten Fensters.

Die folgenden Layout-Bedingungen werden verwendet:

```
layoutmetrics.X.Set(lmRight, lmLeftOf, lmParent, lmRight, 10);
layoutmetrics.Y.Set(lmBottom, lmAbove, lmParent, lmBottom, 10);
layoutmetrics.Width.Absolute(100);
layoutmetrics.Height.Absolute(100);
SetChildLayoutMetrics(*MyChildWindow, layoutMetrics);
```

Beispiel 4

Um ein abhängiges Fenster zu erzeugen, dessen Kanten in einem proportionalen Verhältnis zum übergeordneten Fenster stehen, werden seine Kanten als Prozentanteil des Abstands von den Kanten des übergeordneten Fensters definiert. Insbesondere sind die Ober- und Unterkante des abhängigen Fensters ein Prozentanteil der Unterkante des übergeordneten Fensters und seine linke und rechte Kante ein Prozentanteil von dessen rechter Kante.

Wird die Größe des übergeordneten Fensters verändert, ändert das abhängige Fenster sowohl seine Größe als auch seine absolute Position (das heißt, seine linke und obere Kante ändern sich ebenfalls).

Die folgenden Layoutbedingungen werden verwendet:

```
layoutmetrics.X.Set(lmLeft, lmPercentOf, lmParent, lmRight, 33);
layoutmetrics.Y.Set(lmTop, lmPercentOf, lmParent, lmBottom, 33);
layoutmetrics.Width.Set(lmRight, lmPercentOf, lmParent, lmRight, 66);
layoutmetrics.Height.Set(lmBottom, lmPercentOf, lmParent, lmBottom, 66);
SetChildLayoutMetrics(*MyChildWindow, layoutMetrics);
```

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TLayoutWindow(TWindow* parent, const char far *title = 0, TModule*
              module = 0);
```

Erzeugt ein *TLayoutWindow*-Objekt mit den Angaben für übergeordnetes Fenster, Titel und Bibliotheks-ID.

Destruktor

```
~TLayoutWindow();
```

Löscht Variablen und gibt die Maßangaben und Bedingungen des abhängigen Fensters frei.

Öffentliche Elementfunktionen

GetChildLayoutMetrics

```
bool GetChildLayoutMetrics(TWindow &child, TLayoutMetrics &metrics);
```

Holt die Layoutmaße des abhängigen Fensters.

Siehe auch: *TLayoutMetrics::GetChildMetrics*

Layout

```
void Layout();
```

Bewirkt eine Größenänderung des übergeordneten Fensters und die Neuanordnung der ihm untergeordneten Fenster entsprechend den angegebenen Maßen. *Layout* wird aufgerufen, um Änderungen der Layoutmaße zu implementieren.

RemoveChild LayoutMetrics

```
bool RemoveChildLayoutMetrics(TWindow &child);
```

Löscht die Layoutmaße für ein untergeordnetes Fenster.

SetChildLayoutMetrics

```
void SetChildLayoutMetrics(TWindow &child, TLayoutMetrics &metrics);
```

Setzt die Maße für das Fenster und entfernt dabei alle vorhandenen Maße. Werte werden wie folgt gesetzt:

```
layoutMetrics-X.Absolute(lmLeft, 10);
layoutMetrics-Y.Absolute(lmTop, 10);
layoutMetrics-Width.Set(lmWidth, lmRightOf, GetClientWindow(),
                        lmWidth, -40);
```

```
layoutMetrics-Height.Set(lmHeight, lmRightOf, GetClientWindow(),
                         lmHeight, -40);
```

Dann wird *SetChildLayoutMetrics* aufgerufen, um die Maße mit der Position des untergeordneten Fensters zu verknüpfen:

```
SetChildLayoutMetrics(* MyChildWindow, * layoutMetrics);
```

Geschützte Datenelemente

ClientSize

```
TSize ClientSize;
```

Enthält die Größe des Client-Bereichs.

Geschützte Elementfunktionen

EvSize

```
void EvSize(uint sizeType, TSize& size);
```

Reagiert auf eine Änderung der Fenstergröße durch Aufruf von *Layout* zur Durchführung der Änderung.

Antworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_WM_SIZE	EvSize

Klasse TListBox — listbox.h

Eine *TListBox* ist ein Oberflächen-Objekt, das ein entsprechendes Listenfensterelement in Windows repräsentiert. Eine *TListBox* muß verwendet werden, wenn ein Listenfenster-Dialogelement in einem übergeordneten *TWindow* erzeugt werden soll. Eine *TListBox* kann verwendet werden, um die Kommunikation zwischen einer Anwendung und den Listenfenster-Dialogelementen eines *TDialog* zu erleichtern. Die Elementfunktionen von *TListBox* bedienen auch Instanzen der von *TListBox* abgeleiteten Klasse *TComboBox*. *TListBox* ist eine streamfähige Klasse.

Öffentliche Konstruktoren

Konstruktor

```
TListBox(TWindow* parent, int ID, int x, int y, int w, int h, TModule*
         module = 0);
```

Erzeugt ein Listenfenster-Dialogelement mit den Angaben für die Bibliotheks-ID (*module*) des übergeordneten Fensters (*parent*), für die Position

(*x, y*) relativ zum Ursprungsort des client-Bereichs des übergeord- neten Fensters, für die Breite (*w*) und die Höhe (*h*). Ruft einen *TControl*-Konstruktor auf. Erweitert die Standardeigenschaften des Listenfensters um LBS_STANDARD und stattet es so aus mit:

- einem Rahmen (WS_BORDER)
- einer vertikalen Bildlaufleiste (WS_SCROLL)
- selbsttätiger alphabetischer Sortierung von Listeneinträgen (LBS_SORT)
- Weitermeldung einer Auswahl an das übergeordnete Fenster (LBS_NOTIFY)

Die *TListBox*-Elementfunktionen für Listenfenster mit einfacher Wahlmöglichkeit sind Übernahmen aus *TComboBox* und können auch von Kombinationsfenstern benutzt werden. Diese Elementfunktionen liefern bei Mehrfach-Auswahl -1 zurück.

Konstruktor

```
TListBox(Window* parent, int resId, TModule* module = 0)
```

Erzeugt ein *TListBox*-Objekt zur Verknüpfung mit einem Listenfenster-Dialogelement eines *TDialog*. Ruft den *TControl*-Konstruktor mit ähnlichen Parametern auf. Der *module*-Parameter muß einer zu definierenden Listenfenster-Ressource entsprechen.

Siehe auch: *GetSelIndex, GetSelString, SetSelIndex, SetSelString, TControl::TControl*

Öffentliche Elementfunktionen

AddString

```
virtual int AddString(const char far* str);
```

Erweitert das Listenfenster um *string* und gibt seine Position in der Liste zurück (die erste Position ist 0). Gibt beim Auftreten eines Fehlers einen negativen Wert zurück. Die Listeneinträge werden automatisch sortiert, es sei denn, die Eigenschaft LBS_SORT ist vor dem Generieren aus dem Listenfensterobjekt entfernt worden.

Siehe auch: *TListBox::DeleteString, TListBox::InsertString*

ClearList

```
virtual void ClearList();
```

Löscht alle Einträge in der Liste

DeleteString

```
virtual int DeleteString(int index);
```

Löscht den Eintrag an der in *index* angegebenen Position (beginnend bei 0). *DeleteString* gibt die Anzahl der verbleibenden Listeneinträge zurück oder einen negativen Wert, wenn ein Fehler auftritt.

Siehe auch: *TListBox::AddString, TListBox::InsertString*

DirectoryList

```
virtual int DirectoryList(uint attrs, const char far* fileSpec)
```

Fügt einem Listenfenster eine Liste von Dateinamen hinzu.

FindExactString

```
int FindExactString(const char far* str, int searchIndex) const;
```

Durchsucht die Liste nach einer exakten Übereinstimmung mit der Zeichenkette *str*, beginnend mit der in *searchIndex* angegebenen Zeilennummer. Findet sich nach dem Vergleich der letzten Zeichenkette keine Übereinstimmung, wird die Suche vom Anfang der Liste beginnend fortgesetzt, bis eine gleiche Zeichenkette gefunden wird oder die Liste vollständig durchsucht ist. Sucht vom Anfang der Liste, wenn als *searchIndex* -1 angegeben wurde. Gibt bei erfolgreicher Suche den Index der ersten gefundenen Zeichenkette zurück oder einen negativen Wert beim Auftreten eines Fehlers.

Siehe auch: *TListBox::AddString*, *TListBox::DeleteString*

FindString

```
virtual int FindString(const char far* str, int Index) const;
```

Durchsucht das Listenfenster wie unter *FindExactString* beschrieben, sucht aber nach dem ersten Eintrag, der mit *str* beginnt.

Siehe auch: *TListBox::AddString*, *TListBox::DeleteString*, *TListBox::InsertString*

GetCaretIndex

```
int GetCaretIndex() const;
```

Gibt den Index des Listenelements zurück, das aktuell den Fokus hat. Bei Listenfenstern mit einfacher Wahlmöglichkeit ist der Rückgabewert der Index des ausgewählten Eintrags, falls einer ausgewählt ist.

Siehe auch: *TListBox::SetCaretIndex*

GetCount

```
virtual int GetCount() const;
```

Gibt die Anzahl der Elemente im Listenfenster zurück oder einen negativen Wert, wenn ein Fehler auftritt.

GetHorizontalExtent

```
int GetHorizontalExtent() const;
```

Gibt die Anzahl der Pixel zurück, um die das Listenfenster horizontal verschoben (gescrollt) werden kann.

Siehe auch: *TListBox::SetHorizontalExtent*

GetItemData

```
virtual uint32 GetItemData(int index) const;
```

Gibt den 32-bit Wert des Listenfenster-Elements zurück, das durch *SetItemData* gesetzt ist.

Siehe auch: *TListBox::SetItemData*

GetItemHeight

```
virtual int GetItemHeight(int index) const;
```

Gibt die Höhe des angegebenen Listenfenster-Elements in Pixeln zurück.

Siehe auch: *TListBox::SetItemHeight*

GetItemRect

```
int GetItemRect(int index, TRect& rect) const;
```

Gibt die Abmessungen des Rechtecks zurück, das ein aktuell im Listenfenster angezeigtes Element einschließt.

GetSel

```
bool GetSel(int index) const;
```

Gibt den Auswahlzustand des Listenfensterelements an der durch *index* bezeichneten Position zurück. Gibt **true** zurück, wenn das Element ausgewählt ist und **false**, wenn es nicht ausgewählt ist.

Siehe auch: *TListBox::SetSel*

GetSelCount

```
int GetSelCount() const;
```

Gibt die Anzahl der ausgewählten Elemente in einem Listen- oder Kombifenster mit einfacher oder mehrfacher Wahlmöglichkeit zurück.

GetSelIndex

```
virtual int GetSelIndex() const;
```

Für Listenfenster mit einfacher Wahlmöglichkeit. Liefert den nicht-negativen Index (beginnend mit 0) des aktuell ausgewählten Elements zurück oder einen negativen Wert, wenn kein Element ausgewählt ist.

Siehe auch: *TListBox::SetSelIndex*

GetSelIndexes

```
int GetSelIndexes(int* indexes, int maxCount) const;
```

Für Listenfenster mit mehrfacher Wahlmöglichkeit. Füllt das *indexes*-Array mit den Indizes der ausgewählten Zeichenketten (größtmögliche Anzahl der Zeichenketten: *maxCount*). Gibt die Anzahl der in *indexes* eingegebenen Elemente zurück (-1 für Listenfelder mit einfacher Wahlmöglichkeit und Kombifenster).

Siehe auch: *TListBox::SetSelIndexes*

GetSelString

```
int GetSelString(char far* str, int maxChars) const;
```

Ermittelt die aktuell ausgewählten Elemente und setzt sie in *Strings* ein (größtmögliche Zeichenanzahl: *maxChars*). Jeder Eintrag im *Strings*-Array sollte Platz für *maxChars* Zeichen und eine abschließende Null haben. Gibt für Listenfenster mit einfacher Wahlmöglichkeit die Länge der Zeichenkette zurück, einen negativen Wert beim Auftreten eines Fehlers oder 1, wenn keine Zeichenkette ausgewählt ist. Gibt -1 für Listenfenster mit mehrfacher Wahlmöglichkeit zurück.

Siehe auch: *TListBox::SetSelString*

GetSelStrings

```
int GetSelStrings(char far** strs, int maxCount, int maxChars) const;
```

Ermittelt die Gesamtzahl ausgewählter Elemente für eine Mehrfach-Auswahlliste und kopiert sie in den Puffer. *str* ist ein Array von Zeigern auf Zeichen. Jeder der Zeiger auf die Puffer ist von der Größe *maxChars*. *maxCount* ist die Größe des Arrays.

Siehe auch: *TListBox::SetSelStrings*

GetString

```
virtual int GetString(char far* str, int index) const;
```

Ermittelt das Element an der Position *index* (beginnend bei 0) und gibt es in *str* zurück. *GetString* gibt die Länge der Zeichenkette zurück oder einen negativen Wert beim Auftreten eines Fehlers.

GetStringLen

```
virtual int GetStringLen(int Index) const;
```

Gibt die Länge der Zeichenkette zurück (ohne die abschließende 0) für das Element am Positionsindex *Index*. Gibt einen negativen Wert zurück, wenn ein Fehler auftritt.

GetTopIndex

```
int GetTopIndex() const;
```

Liefert den Index des zuoberst im Listenfenster angezeigten Elements zurück.

Siehe auch: *TListBox::SetTopIndex*

InsertString

```
virtual int InsertString(const char far* str, int index);
```

Setzt *str* an der in *index* angegebenen Position in das Listenfenster ein und gibt die aktuelle Position (beginnend bei 0) des Elements in der Liste zurück. Tritt ein Fehler auf, wird ein negativer Wert zurückgegeben. Die Liste wird nicht neu sortiert. Bei *index* gleich -1 wird die Zeichenkette an das Ende der Liste angefügt.

Siehe auch: *TListBox::AddString*, *TListBox::DeleteString*, *TListBox::FindString*

SetCaretIndex

```
int SetCaretIndex(int index, bool partScrollOk);
```

Setzt den Fokus auf das in *index* angegebene Element. Die Liste wird gegebenenfalls verschoben, so daß ein nicht-sichtbares Element in den angezeigten Ausschnitt gelangt.

Siehe auch: *TListBox::GetCaretIndex*

SetColumnWidth

```
void SetColumnWidth(int width);
```

Setzt die Breite der Elemente im Listenfenster in Pixeln.

SetHorizontalExtent

```
void SetHorizontalExtent(int horzExtent);
```

Setzt die Anzahl der Pixel, um die das Listenfenster horizontal verschoben werden kann.

Siehe auch: *TListBox::GetHorizontalExtent*

SetItemData

```
virtual int SetItemData(int index, uint32 itemData);
```

Setzt den 32-Bit-Wert des Listenfenster-Elements an der angegebenen *index*-Position.

Siehe auch: *TListBox::GetItemData.*

SetItemHeight

```
virtual int SetItemHeight(int index, int height);
```

Setzt die Höhe der Elemente im Listenfenster in Pixeln.

Siehe auch: *TListBox::GetItemHeight*

SetItemRect

```
int SetItemRect(int index, TRect& rect) const;
```

Setzt die Dimensionen des Rechtecks, das ein aktuell im Listenfenster angezeigtes Element umschließt..

SetSel

```
int SetSel(int index, bool select);
```

Wählt ein Element an der in *index* angegebenen Position aus. Für Listenfelder mit Mehrfachauswahl.

Siehe auch: *TListBox::GetSel*

SetSelIndex

```
virtual int SetSelIndex(int index);
```

Für Listenfenster mit einfacher Wahlmöglichkeit. Erzwingt die Auswahl des Elements an der Position *index* (beginnend bei 0).Bei *index* gleich -1 stehen keine Wahlmöglichkeiten im Listenfenster zur Verfügung. *SetSelIndex* liefert bei Auftreten eines Fehlers einen negativen Wert zurück.

SetSelIndexes

```
int SetSelIndexes(int* indexes, int NumSelections, bool shouldSet);
```

Für Listenfenster mit Mehrfachwahl. Selektiert/deselektiert die Zeichenketten in dem verbundenen Listenfenster an den im *indexes*-Array angegebenen Indizes. Ist *shouldSet* **true**, werden die indizierten Zeichenketten ausgewählt und hervorgehoben; bei *shouldSet* **false** werden Hervorhebung und Auswahl aufgehoben. Gibt die Anzahl der erfolgreich selektierten oder deselektierten Zeichenketten zurück (-1 für Listenfenster mit Einfachwahl und Kombifenster). Wenn *NumSelections* kleiner als 0 ist, werden alle Zeichenketten selektiert oder deselektiert und ein negativer Wert zurückgegeben, wenn ein Fehler auftritt.

SetSelItemRange

```
int SetSelItemRange(bool select, int first, int last);
```

Wählt die Elemente im Bereich von *first* bis *last*.

SetSelString

```
int SetSelString(const char far* str, int searchIndex);
```

Für Listenfenster mit einfacher Wahlmöglichkeit. Erzwingt die Auswahl des ersten Elements nach der in *SearchIndex* angegebenen Position (beginnend bei 0), das mit dem in *str* angegebenen Text beginnt. Wenn *SearchIndex* -1 ist, wird die ganze Liste durchsucht, beginnend beim ersten Element. *SetSelString* gibt die Position des neu ausgewählten Elements zurück, -1 beim Auftreten eines Fehlers.

SetSelStrings

```
int SetSelStrings(const char far** prefixes, int numSelections,
                  bool shouldSet);
```

Für Listenfenster mit Mehrfachwahl. Wählt die Zeichenketten in dem verbundenen Listenfenster, die mit den im *prefixes*-Array angegebenen Präfixen anfangen. Für jede Zeichenkette beginnt der Suchvorgang am Anfang der Liste und wird fortgesetzt, bis ein passender Eintrag gefunden oder die Liste vollständig durchsucht wurde. Wenn *shouldSet* **true** ist, werden die passenden Zeichenketten ausgewählt und hervorgehoben; ist *shouldSet* **false**, werden Hervorhebung und Auswahl der passenden Zeichenketten aufgehoben. Gibt die Anzahl der erfolgreich selektierten oder deselektierten Zeichenketten zurück (-1 für Listenfenster mit Einfachwahl und Kombifenster). Ist *numSelections* kleiner als 0, werden alle Zeichenketten selektiert oder deselektiert; ein negativer Wert wird beim Auftreten eines Fehlers zurückgegeben.

SetTabStops

```
bool SetTabStops(int numTabs, int far* tabs);
```

Setzt Tabulatorstopps. *numTabs* ist die Anzahl der Tabstopps. *tabs* ist das Array der ganzen Zahlen, die die Tabulatorpositionen darstellen.

SetTopIndex

```
int SetTopIndex(int index);
```

Setzt *index* auf das erste Element, das oben im Listenfenster angezeigt ist.

Siehe auch: *TListBox::GetTopIndex*

Transfer

```
uint Transfer(void *buffer, TTransferDirection direction);
```

Überträgt die Elemente und die Auswahl des Listenfensters in einen oder aus einem Übertragungsspeicher, wenn *tdSetData*, bzw. *tdGetData* als *direction* übergeben wird. Es wird davon ausgegangen, daß *buffer* auf einen Zeiger auf eine *TListBoxData*-Struktur zeigt.

Transfer, das die virtuelle Elementfunktion *TWindow* umgeht, gibt die Größe von *TListBoxData* zurück (den Zeiger, nicht die Struktur). Zur Ermittlung der Größe ohne Übertragung von Daten ist *tdSizeData* als *direction* zu übergeben.

Im Übertragungsspeicher muß ein Zeiger auf diese Strukturen verwendet werden. Weder können Kopien der Strukturen in den Übertragungsspeicher eingebettet werden, noch können diese Strukturen selbst als Übertragungsspeicher verwendet werden.

Siehe auch: *TListBoxData, TWindow::Transfer*

Geschützte Elementfunktionen

GetClassName

```
char far* GetClassName();
```

Gibt den Namen der Windows-Registrierungsklasse von *TListBox* zurück, "LISTBOX".

Struktur TListBoxData listbox.h

TListBoxData ist eine Struktur, die zum Übertragen des Inhalts eines Listenfensters verwendet wird.

Öffentliche Datenelemente

ItemDatas

```
TDwordArray* ItemDatas;
```

Enthält alle Elementdaten für jedes Element im Listenfenster im uint32-Format.

SelCount

```
int SelCount;
```

Enthält die Anzahl der ausgewählten Elemente.

SelIndices

```
TIntArray* SelIndices;
```

Enthält die Indizes aller ausgewählten Zeichenketten in einem Listenfenster mit mehrfacher Wahlmöglichkeit.

SelStrings

```
TStringArray* SelStrings;
```

Zeiger auf ein Array der Zeichenketten, die auszuwählen sind, wenn Daten in das Listenfenster übertragen werden. Wenn Daten aus dem Listenfenster übertragen werden, gibt *SelStrings* die aktuelle Auswahl zurück.

Strings

```
TStringArray* Strings;
```

Zeiger auf ein Array von Zeichenketten, die in ein *TListBox*-Listenfenster zu übertragen sind.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TListBoxData();
```

Konstruiert *Strings* und *SelStrings*. Initialisiert *SelCount* mit 0.

Destruktor

```
~TListBoxData();
```

Löscht den an *Strings* und *SelStrings* zugewiesenen Speicherplatz.

Öffentliche Elementfunktionen

AddString

```
void AddString(const char *str, bool isSelected = false);
```

Fügt *Strings* die angegebene Zeichenkette hinzu. Fügt die Zeichenkette zu *SelStrings* hinzu und erhöht *SelCount*, wenn *IsSelected* **false** ist.

AddStringItem

```
void AddStringItem(const char* str, uint32 itemData,
                   bool isSelected = false);
```

Fügt dem *Strings*-Array eine Zeichenkette hinzu, wählt sie auf Wunsch aus, und fügt die Element-Daten dem *ItemDatas*-Array hinzu.

Clear

```
void Clear();
```

Setzt das Listenfenster zurück, indem es die Arrays *ItemDatas* und *Strings* leert und *ResetSelections* aufruft.

Siehe auch: *TListBoxData::Strings, TListBoxData::ItemDatas, TListBoxData::ResetSelections*

GetItemDatas

```
TDwordArray& GetItemDatas();
```

Liefert einen Zeiger auf das Array *ItemDatas* zurück.

Siehe auch: *TListBoxData::ItemDatas*

GetSelIndices

```
TIntArray& GetSelIndices();
```

Liefert einen Zeiger auf das Array *SelIndices* zurück.

Siehe auch: *TListBoxData::SelIndices*

GetSelString

```
void GetSelString(char far* buffer, int bufferSize, int index = 0) const;
```

Lokalisiert die Zeichenkette an dem in *SelStrings* angegebenen *index* und kopiert sie in *buffer*. *bufferSize* enthält die abschließende 0.

GetSelStringLength

```
int GetSelStringLength(int index = 0) const;
```

Gibt die Länge (ohne die abschließende 0) der Zeichenkette am angegebenen *index* in *SelStrings* zurück.

GetStrings

```
TStringArray& GetStrings();
```

Gibt einen Zeiger auf das Array *Strings* zurück.

Siehe auch: *TListBoxData::Strings*

ResetSelections

```
void ResetSelections();
```

Entfernt alle Zeichenketten aus *SelStrings* und setzt *SelCounts* auf 0.

Select

```
void Select(int index);
```

Wählt die Zeichenkette an dem angegebenen Index aus.

SelectString

```
void SelectString(const char far* str);
```

Fügt *str* zu *SelStrings* hinzu und erhöht *SelCount*.

Klasse TListView listview.h

TListView ist aus *TListBox* und *TView* abgeleitet und stellt Views für Listenfenster zur Verfügung. Eine Beschreibung von View-Funktionen findet sich unter *TView*, unter *TListBox* eine Beschreibung von Listenfensterfunktionen.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TListView(TDocument& doc, TWindow* parent = 0);
```

Erzeugt ein *TListView*-Objekt, das an das angegebene Dokument und das angegebene übergeordnete Fenster gebunden ist. Setzt *Attr.Accel.Table* auf IDA_LISTVIEW, um die Edit-View zu identifizieren. Setzt *Attr.Style* auf WS_HSCROLL | LBS_NOINTEGRALHEIGHT. Setzt *TView::ViewMenu* auf das neue *TMenuDescr* für diese View.

Destruktor

```
~TListView();
```

Nachdem er überprüft hat, ob eine offene View existiert, entfernt der Destruktor das *TListView*-Objekt.

Öffentliches Datenelement

DirtyFlag

```
bool DirtyFlag;
```

Ist ungleich 0, wenn sich Daten in der Listen-View geändert haben; andernfalls 0.

Öffentliche Elementfunktionen

CanClose

```
bool CanClose();
```

Überprüft, ob alle untergeordneten Views geschlossen werden können, bevor die aktuelle View geschlossen wird. Wenn eine untergeordnete View 0 zurückgibt, gibt *CanClose* 0 zurück und bricht den Vorgang ab. Ruft *TDocManager::FlushDoc* auf, wenn alle abhängigen Views-Werte ungleich 0 zurückgeben.

Siehe auch: *TView::CanClose, TDocManager::FlushDoc*

Create

```
virtual bool Create();
```

Überschreibt *TWindow::Create* und ruft *TEditSearch::Create* auf, um das Fenster für die View zu erzeugen. Ruft *GetDocPath* auf, um festzustellen, ob die Datei neu ist oder bereits Daten enthält. Ruft *LoadData* auf, wenn Daten vorhanden sind, das der View die Daten hinzufügt. Kann das Fenster für die View nicht erzeugt werden, löst *Create* eine *TXInvalidWindow*-Exception aus.

GetViewName

```
LPCSTR GetViewName();
```

Überschreibt die virtuelle Funktion von *TView* und gibt den deskriptiven Namen der Klasse zurück (*StaticName*).

Siehe auch: *TView::GetViewName*

GetWindow

```
TWindow* GetWindow();
```

Überschreibt die virtuelle Funktion von *TView* und liefert das Listen-View-Objekt als *TWindow* zurück.

Siehe auch: *TView::GetWindow*

SetDocTitle

```
bool SetDocTitle(LPCSTR docname, int index);
```

Überschreibt die virtuelle *TView*-Funktion und speichert den Dokumentnamen. Dieser Name wird durch die Fensterhierarchie nach oben weitergegeben, bis ein *TFrameWindow*-Objekt die Daten annimmt und sie in seiner Überschrift anzeigt.

Siehe auch: *TView::SetDocTitle*

StaticName

```
static LPCSTR StaticName();
```

Überschreibt die *TView*-Funktion und gibt eine konstante Zeichenkette zurück, »ListView«. Diese Information wird im Auswahlfenster der Benutzeroberfläche angezeigt.

Siehe auch: *TView::GetViewName*

Geschützte Datenelemente

MaxWidth

```
int MaxWidth;
```

Enthält die größte horizontale Ausdehnung (die Pixelanzahl, um die die View horizontal verschoben werden kann).

Origin

```
long Origin;
```

Enthält die Dateiposition am Anfang der Anzeige.

Geschützte Elementfunktionen

CmEditAdd

```
void CmEditAdd();
```

Reagiert automatisch auf die CM_LISTADD-Botschaft. Holt die Länge der eingegebenen Zeichenkette und ruft *InsertString* auf, um die Zeichenkette in die Listen-View einzufügen. Setzt das Datenelement *DirtyFlag* auf **true.**

CmEditClear

```
void CmEditClear();
```

Reagiert automatisch auf eine Menüauswahl mit einer Menü-ID von CM_EDITCLEAR durch Aufruf von *TListBox::Clear* zum Löschen der Listen-View.

CmEditCopy

```
void CmEditCopy();
```

Reagiert automatisch auf eine Menüauswahl mit einer Menü-ID von CM_EDITCOPY durch Aufruf von *TListBox::Copy* zum Kopieren des ausgewählten Textes ins Ablagefeld.

CmEditCut

```
void CmEditCut();
```

Reagiert automatisch auf eine Menüauswahl mit einer Menü-ID von CM_EDITCUT durch Aufruf von *CmEditCopy* und *CmEditDelete* zum Löschen einer Text-Zeichenkette aus der Listen-View. Setzt das Datenelement *DirtyFlag* auf **true.**

CmEditDelete

```
void CmEditDelete();
```

Reagiert automatisch auf eine Menüauswahl mit einer Menü-ID von CM_EDITDELETE durch Aufruf von *TListBox::DeleteSelection*.

CmEditItem

```
void CmEditItem();
```

Reagiert automatisch auf eine CM_LISTEDIT-Botschaft durch Holen und Einfügen des Eingabetextes in die Listen-View. Setzt das *DirtyFlag* auf einen Wert ungleich 0, um anzuzeigen, daß sich die View geändert hat und nicht gespeichert wurde.

CmEditPaste

```
void CmEditPaste();
```

Reagiert automatisch auf eine Menüauswahl mit einer Menü-ID von CM_EDITPASTE durch Aufruf von *TListBox::Paste*.

CmEditUndo

```
void CmEditUndo();
```

Reagiert automatisch auf eine Menüauswahl mit einer Menü-ID von CM_EDITUNDO durch Aufruf von *TListBox::Undo*.

CmSelChange

```
void CmSelChange();
```

Reagiert automatisch auf eine LBN_SELCHANGE-Botschaft (die eine Änderung des Inhalts einer Listen-View anzeigt) durch Aufruf von *DefaultProcessing*.

EvGetDlgCode

```
uint EvGetDlgCode(MSG far* msg);
```

Überschreibt die Reaktion von *TWindow* auf eine WM_GETDLGCODE-Botschaft (eine Eingabeprozedur, die zu einem Dialogelement - ausgenommen Markierungsfelder - gehört) durch Aufruf von *DefaultProcessing*. Der Parameter *msg* zeigt die Art der Botschaft an, die an den Dialogfensterverwalter gesendet wird, z.b. eine Dialogelement-Botschaft oder einen Befehl.

EvGetDlgCode gibt einen Code zurück, der anzeigt, wie die Botschaft des Listenfenster-Dialogelements zu behandeln ist.

Siehe auch: *TButton::EvGetDlgCode, TWindow::DefaultProcessing, Dialogelement-Botschaftskonstanten DLGC_xxxx*

LoadData

```
bool LoadData(int top, int sel);
```

Liest die View aus dem Stream und schließt die Datei. Gibt **true** zurück, wenn das Laden der View erfolgreich war.

Löst eine *xmsg*-Exception aus und zeigt die Fehlermeldung "TListView initial read error ", wenn die Datei nicht gelesen werden kann. **false** zurück, wenn die View nicht geladen werden kann.

SetExtent

```
void SetExtent(LPSTR str);
```

Setzt die maximale horizontale Ausdehnung für das Fenster der Listen-View.

VnCommit

```
bool VnCommit(bool force);
```

VnCommit überweist in der View vorgenommene Änderungen an das Dokument. Bei *force* ungleich 0 werden alle Daten im Dokument gespeichert, auch wenn sie unverändert sind.

Siehe auch: *TListView::vnRevert, vnxxxx*-View-Benachrichtigungs-Konstanten

VnDocClosed

```
bool VnDocClosed(int omode);
```

VnDocClosed zeigt an, daß das Dokument geschlossen wurde.

Siehe auch: *vnxxxx* View-Benachrichtigungs-Konstanten

VnIsDirty

```
bool VnIsDirty();
```

VnIsDirty gibt einen Wert ungleich 0 zurück, wenn Datenänderungen in der View nicht im Dokument gespeichert wurden; gibt andernfalls 0 zurück.

Siehe auch: *vnxxxx* View-Benachrichtigungs-Konstanten

VnIsWindow

```
bool VnIsWindow(HWND hWnd);
```

VnIsWindow liefert einen Wert ungleich 0 zurück, wenn das von *hWnd* übergebene Fenster-Handle mit dem des Anzeigefensters der View identisch ist.

Siehe auch: *vnxxxx* View-Benachrichtigungs-Konstanten

VnRevert

```
bool VnRevert(bool clear);
```

VnRevert zeigt an, ob in der View vorgenommene Änderungen zu löschen sind und die View mit den Daten aus dem Dokument wiederherzustellen ist. Wenn *clear* ungleich 0 ist, werden die Daten aus dem Dokument nicht an die View übergeben, sondern gelöscht.

Siehe auch: *TListView::vnCommit*

Antworttabelleneinträge

Eintrag	Elementfunktion
EV_COMMAND(CM_LISTUNDO, CmEditUndo	CmEditUndo
EV_COMMAND(CM_LISTCUT, CmEditCut)	CmEditCut
EV_COMMAND(CM_LISTCOPY, CmEditCopy)	CmEditCopy
EV_COMMAND(CM_LISTPASTE, CmEditPaste)	CmEditPaste
EV_COMMAND(CM_LISTCLEAR, CmEditClear)	CmEditClear
EV_COMMAND(CM_LISTDELETE, CmEditDelete)	CmEditDelete
EV_COMMAND(M_LISTADD, CmEditAdd)	CmEditAdd
EV_COMMAND(CM_LISTEDIT, CmEditItem)	CmEditItem
EV_WM_GETDLGCODE	EvGetDlgCode
EV_NOTIFY_AT_CHILD(LBN_DBLCLK, CmEditItem)	CmEditItem
EV_NOTIFY_AT_CHILD(LBN_SELCHANGE, CmSelChange)	CmSelChange
EV_VN_DOCCLOSED	VnDocClosed
EV_VN_ISWINDOW	VnIsWindow
EV_VN_ISDIRTY	VnIsDirty
EV_VN_COMMIT	VnCommit
EV_VN_REVERT	VnRevert

Struktur TLocaleString locale.h

Die Struktur *TLocaleString* bietet Unterstützung für lokalisierte Registrierparameter und definiert einen lokaliserbaren Ersatz für Strings vom Typ **char*.** Diese Strings, die sowohl OLE-fähige als auch nicht-OLE-fähige Objekte für den Anwender beschreiben, können in jeder gewünschten Sprache zur Verfügung gestellt werden. Die Struktur unterstützt gleichermaßen das Dok/View-Modell von ObjectWindows wie OLE-fähige ObjectComponents-Anwendungen.Die öffentlichen Elementfunktionen, die Informationen über die Sprache des Anwenders, die native Sprache der Anwendung sowie eine Beschreibung der für die Lokalisierung markierten Strings zur Verfügung stellen, vereinfachen das Übersetzen und Vergleichen von Strings in einer gegebenen Sprache.

TLocaleString verwendet vom Benutzer einzugebende Präfixe, die festlegen, welche Art von Strings übersetzt werden sollen. Jedem Präfix muß ein gültiger Ressourcenbezeichner folgen (ein C-Standardbezeichner). Die folgende Tabelle verzeichnet die Präfixe, die *TLocaleString* zum Lokalisieren von Strings verwendet. Zu jedem Präfix ist ein Beispieleintrag angeführt.

Präfix	Beschreibung
@TXY	Der String ist eine Reihe von Zeichen, die als Ressourcen-ID interpretiert werden, und auf die nur aus einer Ressourcendatei zugegriffen wird. Er wird nie direkt verwendet.
#1045	Der String ist eine Reihe von Ziffern, die als Ressourcen-ID interpretiert werden, und auf die nur aus einer Ressourcendatei zugegriffen wird. Er wird nie direkt verwendet.
!MyWindow	Der String wird übersetzt, wenn er nicht in der nativen Sprache der Anwendung verwendet wird; in diesem Fall wird er direkt verwendet.

Weitere Informationen über das Lokalisieren von Strings finden Sie im *ObjectWindows Programmierhandbuch* im Abschnitt über das Lokalisieren von Symbolnamen.

Öffentliche Elementfunktionen

Compare

```
int Compare(const char far* str, TLangId lang);
```

Compare vergleicht einen *TLocaleString* mit einem anderen String und verwendet dazu die angegebene Sprache (lang). Es setzt einen Standard-Stringvergleich ein und das sprachspezifische Kollationsschema. Es gibt einen der folgenden Werte zurück.

Rückgabewert	Bedeutung
0	Es besteht keine Übereinstimmung zwischen beiden Strings.
1	Dieser String ist länger als der andere String.
-1	Dieser String ist kürzer als der andere String.

GetSystemLangId

```
static TLangId GetSystemLangId();
```

Gibt die Sprachkennung des Systems zurück; diese kann mit der *UserLangId* übereinstimmen.

Siehe auch: *TLocaleString::GetUserLangId*

GetUserLangId

```
static TLangId GetUserLangId();
```

Gibt die Kennung der Benutzersprache zurück. Bei Einbenutzersystemen ist das *LangSysDefault*. Die Sprachkennung ist eine vordefinierte Zahl, die eine Grundsprache und einen Dialekt bezeichnet.

Siehe auch: *TLocaleString::GetSystemLangId*

IsNativeLangId

```
static int IsNativeLangId(TLangId lang);
```

Gibt **true** zurück, wenn *lang* die native Systemsprache ist.

operator const char*

```
operator const char* ();
```

Liefert den aktuellen Zeichen-String in der Übersetzung zurück.

operator =

```
void operator = (const char* str);
```

Weist diesem Locale-String den String (*str*) zu.

Translate

```
const char* Translate(TLangId lang);
```

Übersetzt den String in die angegebene Sprache. Bei der Auswahl der Übersetzungssprache setzt *Translate* die Prioritäten gemäß folgender Liste:

1. Grundsprache und Dialekt
2. Grundsprache ohne Dialekt
3. Grundsprache und anderer Dialekt
4. Ursprungssprache des Strings selbst.

Liefert 0 zurück, wenn es den String nicht übersetzen kann. (Kann nur bei Verwendung der Präfixe @ oder # vorkommen; andernfalls zeigt das Präfix ! an, daß der folgende String in der Ursprungssprache steht).

Siehe auch: *Typdefinition TLangId, Konstanten LangXxxx_ID*

Klasse TLookupValidator — validate.h

TLookupValidator, eine streamfähige Klasse, vergleicht die von einem Benutzer eingegebene Zeichenkette mit einer Liste gültiger Werte. *TLookupValidator* ist ein abstrakter Typ von Prüfungsobjekten, von denen anwendbare Nachschlageprüfungsobjekte abgeleitet werden können. Es kann niemals eine Instanz von *TLookupValidator* erzeugt werden. Zum Erzeugen eines Nachschlageprüfungstyps muß eine Liste gültiger Elemente festgelegt und die *Lookup*-Methode umgangen werden, damit **true** nur dann zurückgegeben wird, wenn eine Benutzereingabe mit einem Element dieser Liste übereinstimmt. Ein Beispiel einer anwendbaren Ableitung von *TLookupValidator* ist *TStringLookupValidator*.

Öffentliche Konstruktoren

Konstruktor

```
TLookupValidator();
```

Erzeugt ein *TLookupValidator*-Objekt.

Öffentliche Elementfunktionen

IsValid

```
bool IsValid(const char far* str);
```

IsValid umgeht die virtuelle Funktion von *TValidator* und ruft *Lookup* auf, um die Zeichenkette *str* in der Liste gültiger Eingabewerte zu finden. *IsValid* liefert **true** zurück, wenn *Lookup* **true** zurückliefert. Dies bedeutet, daß

Lookup str in seiner Liste gefunden hat; andernfalls wird **false** zurückgegeben.

Lookup

```
virtual bool Lookup(const char far* str);
```

Sucht nach der Zeichenkette *str* in der Liste gültiger Einträge und gibt **true** zurück, wenn *str* gefunden wird, andernfalls **false**. *Lookup* ist die in *TValidatorLookup* eingesetzte, abstrakte Methode. Sie gibt immer **false** zurück. Ihre Nachkommen als Nachschlageprüfungstypen müssen *Lookup* außer Kraft setzen, um eine Suche auf der Grundlage der tatsächlichen Liste gültiger Elemente durchzuführen.

Klasse TMDIChild mdichild.h

TMDIChild definiert das grundlegende Verhalten aller untergeordneten MDI-Fenster. Untergeordnete Fenster können innerhalb des Client-Bereichs des übergeordneten Fensters erzeugt werden. Da untergeordnete Fenster innerhalb eines übergeordneten Fenster bestehen und auf dessen Abmessungen beschränkt sind, muß das übergeordnete Fenster vor dem untergeordneten erzeugt werden. Ein Dialogfenster ist beispielsweise ein Fenster, das untergeordnete Fenster besitzt, die oft als Dialogfenster-Elemente bezeichnet werden.

Untergeordnete und verwendbare MDI-Klassen müssen aus *TMDIChild* abgeleitet werden. Solche abgeleiteten MDI-Klassen können Tastaturbedienbarkeit, Fokusbehandlung und Symbolunterstützung von *TFrameWindow* erben. *TMDIChild* ist eine streamfähige Klasse.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TMDIChild(TMDIClient &parent, const char far* title = 0,
          TWindow* clientWnd = 0, bool shrinkToClient = false, TModule*
          module = 0);
```

Erzeugt - unter Verwendung derAngaben für *title*, Client-Fenster (*client Wnd*) und Instanz (*inst*) - ein MDI-Fenster, das dem in *parent* festgelegten MDI-Client-Window untergeordnet ist. Ruft den *TFrameWindow* Basisklassen-Konstruktor auf und legt *parent*, *title, clientWnD* und *inst* fest, und zeigt an, daß die Größe des untergeordneten Fensters nicht angepaßt werden soll. Ruft den Basisklassenkonstruktor von *TWindow* auf und legt *parent*, *title* und *inst* fest. Die Fensterattribute werden dann zur Einbindung von WS_VISIBLE, WS_CHILD, WS_CLIPSIBLINGS, WS_CLIPCHILDREN, WS_SYSMENU, WS_CAPTION, WS_THICKFRAME, WS_MINIMIZEBOX und WS_MAXIMIZEBOX angepaßt. Die Abmessungen des Fensters werden auf seine Vorgabewerte gesetzt.

Konstruktor

```
TMDIChild(HWND hWnd, TModule* module = 0);
```

Erzeugt ein untergeordnetes MDI-Fensterobjekt aus einem bereits vorhandenen Fenster, das durch *hWnd* festgelegt ist. Der Basisklassenkonstruktor von *TFrameWindow* wird unter Angabe von *hWnd* wie auch der festgelegten *inst* aufgerufen. Der Basisklassenkonstruktor von *TWindow* wird aufgerufen, unter Angabe der *hWnd*- und *inst*-Parameter.

Destruktor

```
~TMDIChild();
```

Entfernt das untergeordnete MDI-Fensterobjekt.

Öffentliche Elementfunktionen

Destroy

```
void Destroy(retVal = 0);
```

Zerstört das an *TMDIChild* gebundene Oberflächenobjekt. Ruft für jedes untergeordnete Fenster in der Liste *EnableAutoCreate* auf. Dadurch werden alle untergeordneten Fenster neu erzeugt, wenn das übergeordnete Fenster neu erzeugt wird.

Siehe auch: *TWindow::EnableAutoCreate*

EnableWindow

Überschreibt die virtuelle Funktion von *TWindow*. Aktiviert ein untergeordnetes Fenster.

PreProcessMsg

```
bool PreProcessMsg (MSG & msg);
```

Führt die Vorverarbeitung (preprocessing) von Fensterbotschaften für das untergeordnete MDI-Fenster durch. Die *TMDIClient::PreProcessMsg*-Elementfunktion des übergeordneten Client-Fensters wird aufgerufen, wenn die Verarbeitung von Tastatureingaben festgelegt ist. In diesem Fall ist der Rückgabewert **true**. Andernfalls wird *TFrameWindow::PreProcessMsg* aufgerufen und gibt den Rückgabewert dieser Elementfunktion zurück.

Siehe auch: *TMDIClient::PreProcessMsg, TFrameWindow::PreprocessMsg*

ShowWindow

```
bool ShowWindow(int cmdShow);
```

Überschreibt die virtuelle Funktion von *TWindow*. Zeigt ein untergeordnetes Fenster gemäß dem Wert von *cmdShow* an.

Geschützte Elementfunktionen

DefWindowProc

```
LRESULT DefWindowProc (uint msg, WPARAM wParam, LPARAM lParam);
```

Überschreibt *TWindow::DefWindowProc* und stellt so eine Standardverarbeitung für alle eingehenden Botschaften zur Verfügung, die nicht vom untergeordneten Fenster selbst verarbeitet werden. Außerdem verarbeitet *DefWindowProc* die folgenden Botschaften: WM_CHILDACTIVATE, WM_GETMINMAXINFO, WM_MENUCHAR, WM_MOVE, WM_SETFOCUS, WM_SIZE und WM_SYSCOMMAND.

Siehe auch: *::DefMDIChildProc, TWindow::DefWindowProc*

EvMDIActivate

```
void EvMDIActivate(HWND hWndActivated, HWND hWndDeactivated);
```

Weist ein Client-Fenster an, ein untergeordnetes MDI-Fenster zu aktivieren oder zu deaktivieren, und teilt dies dem untergeordneten Fenster durch Senden einer Botschaft mit.

EvNCActivate

```
void EvNCActivate(bool activate);
```

Reagiert auf eine Anforderung zum Wechsel einer Titelleiste oder eines Symbols.

PerformCreate

```
void PerformCreate(int menuOrId),
```

Erzeugt das an das untergeordnete MDI-Fenster gebundene Oberflächenelement. Löst eine *TWindow::TXInvalidChildWindow*-Exception aus, wenn kein übergeordnetes Fenster festgelegt wurde. Ruft andernfalls *::SendMessage* auf, um dem übergeordneten MDI-Client-Fenster die Erzeugung des Oberflächenelements für das untergeordnete Fenster mitzuteilen. Untergeordnete MDI-Fenster können keine Menüs haben, daher wird der *menuOrId*-Parameter ignoriert.

Antworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_WM_MDIACTIVATE	EvMDIActivate
EV_WM_NCACTIVATE	EvNCActivate

Klasse TMDIClient — mdi.h

Client-Fenster (durch ein *TMDIClient*-Objekt repräsentiert) von Multidokumentschnittstellen (MDI, Multiple Document Interface) verwalten die untergeordneten MDI-Fenster eines *TMDIFrame*-Fensters. *TMDIClient* ist eine streamfähige Klasse.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TMDIClient(TModule* module = 0);
```

Erzeugt ein MDI-Client-Fensterobjekt durch Aufruf des Basisklassenkonstruktors von *TWindow*, dem ein leeres übergeordnetes Fenster übergeben wird, ein leerer Titel sowie die festgelegte Bibliotheks-ID. Setzt *Attr.Id* auf den voreingestellten Client-Fenster-Bezeichner (IDW_MDICLIENT) und setzt *Attr.Style* auf Einbindung von MDIS_ALLCHILDSTYLES,

WS_GROUP, WS_TABSTOP, WS_CLIPCHILDREN, WS_VSCROLL und WS_HSCROLL. Initialisiert das *ClientAttr*-Datenelement und setzt sein *idFirstChild*-Element auf IDW_FIRSTMDICHILD.

Siehe auch: *TWindow::TWindow*

Destruktor

```
~TMDIClient();
```

Gibt die Struktur *ClientAttr* frei.

Siehe auch: *TWindow::~TWindow*

Öffentliches Datenelement

ClientAttr

```
LPCLIENTCREATESTRUCT ClientAttr;
```

ClientAttr enthält einen Zeiger auf die Attribute des MDI-Client-Fensters.

Öffentliche Elementfunktionen

ArrangeIcons

```
virtual void ArrangeIcons();
```

Ordnet die Symbole der untergeordneten MDI-Fenster unten im MDI-Client-Fenster an.

CascadeChildren

```
virtual void CascadeChildren();
```

Ordnet alle untergeordneten MDI-Fenster, die nicht als Symbol dargestellt werden, in dem MDI-Client-Fenster in angepaßter Größe an. Die untergeordneten Fenster überlappen, die Titelleisten sind jedoch sichtbar.

CloseChildren

```
virtual bool CloseChildren();
```

Ruft *CanClose* für jedes der untergeordneten MDI-Fenster dieses MDI-Clients auf. Gibt **true** zurück, wenn alle Fenster geschlossen sind, andernfalls **false**.

Siehe auch: *TWindow::CanClose*

Create

```
bool Create();
```

Erzeugt das zu dem MDI-Client-Fenster gehörende Oberflächenelement. Setzt zunächst das Menü-Fenster des untergeordneten Fensters in *ClientAttr* auf das untergeordnete Menü-Fenster des übergeordneten Rahmenfensters und ruft dann *TWindow::Create* auf.

Siehe auch: *TWindow::Create*, *TFrameWindow::GetMenuDescr*

CreateChild

```
virtual TWindow* CreateChild();
```

Überschreibt die von *TWindow* definierte Elementfunktion. Erzeugt eine neues untergeordnetes MDI-Fenster durch Aufruf von *InitChild* und *Create*. Gibt einen Zeiger auf das neue untergeordnete MDI-Fenster zurück.

Siehe auch: *TMDIClient::InitChild*, *TModule::MakeWindow*, *TWindow::Create*

GetActiveMDIChild

```
TMDIChild *GetActiveMDIChild();
```

GetActiveMDIChild zeigt auf das aktive untergeordnete MDI-Fenster von *TMDIClient*. Wird vom untergeordneten Fenster in seiner Botschaftsantwort-Elementfunktion *EvMDIActivate* gesetzt. Die Konstruktoren von *TMDIClient* initialisieren *GetActiveChild*.

InitChild

```
virtual TMDIChild *InitChild();
```

Erzeugt eine Instanz von *TWindow* als untergeordnetes MDI-Fenster und liefert einen Zeiger auf dieses zurück. Untergeordnete Fenster müssen mit einem MDI-Client als übergeordnetem Fenster erzeugt werden. Um eine Instanz einer abgeleiteten Klasse untergeordneter MDI-Fenster zu erzeugen, ist diese Elementfunktion in der abgeleiteten MDI-Fensterklasse neu zu definieren. Beispiel:

```
PTWindowsObject TMyMDIClient::InitChild()
  {
     return new TMyMDIChild(this, "");
  }
```

Siehe auch: *TMDIClient::CreateChild*

PreProcessMsg

```
bool PreprocessMsg(MSG &msg);
```

Stammt die festgelegte *msg* von WM_KEYDOWN oder WM_SYSKEYDOWN, werden die Tastenkürzel für den MDI-Client übersetzt.

Siehe auch: *TWindow::PreprocessMsg*

TileChildren

```
virtual void TileChildren(int title = MDITILE_VERTICAL);
```

Ordnet alle untergeordneten MDI-Fenster, die nicht als Symbole dargestellt sind, in dem MDI-Client-Fenster an und paßt ihre Größe an. Die untergeordneten Fenster nehmen die gesamte Fläche des Client-Fensters ein, ohne zu überlappen.

Geschützte Elementfunktionen

CmArrangeIcons

```
void CmArrangeIcons();
```

Reagiert auf eine Menüauswahl mit einer ID von CM_ARRANGEICONS durch Aufruf von *ArrangeIcons*.

Siehe auch: *TDMIClient::ArrangeIcons*

CmCascadeChildren

```
void CmCascadeChildren();
```

Reagiert auf eine Menüauswahl mit einer ID von CM_CASCADECHILDREN durch Aufruf von *CascadeChildren*.

Siehe auch: *TDMIClient::CascadeChildren*

CmChildActionEnable

```
void CmChildActionEnable(TCommandEnabler& commandEnabler);
```

Wenn es untergeordnete MDI-Fenster gibt, aktiviert *CmChildActionEnable* alle in ihnen zur Verfügung stehenden Aktionsfensterelemente.

CmCloseChildren

```
void CmCloseChildren();
```

Reagiert auf eine Menüauswahl mit einer ID von CM_CLOSECHILDREN durch Aufruf von *CloseChildren*.

Siehe auch: *TMDIClient::CloseChildren*

CmCreateChild

```
void CmCreateChild();
```

Reagiert auf eine Menüauswahl mit einer ID von CM_CREATECHILD durch Aufruf von *CreateChild*, um ein neues untergeordnetes Fenster zu erzeugen.

Siehe auch: *TMDIClient::CreateChild*

CmTileChildren

```
void CmTileChildren();
```

Reagiert auf eine Menüauswahl mit einer ID von CM_TILECHILDREN durch Aufruf von *TileChildren*.

Siehe auch: *TMDIClient::TileChildren*

CmTileChildrenHoriz

```
void CmTileChildrenHoriz();
```

Reagiert auf eine Menüauswahl mit einer ID von CM_TILECHILDREN durch Aufruf von *TileChildren* und übergibt das Titel-Flag des untergeordneten MDI-Fensters als MDITILE_HORIZONTAL.

EvMDICreate

```
LRESULT EvMDICreate(MDICREATESTRUCT far& createStruct);
```

Fängt die Botschaft WM_MDICREATE ab, die gesendet wird, wenn untergeordnete MDI-Fenster erzeugt werden. Wenn MDIS_ALLCHILDSTYLES zu den Eigenschaften des Client-Fensters gehört, ändert *EvMDICreate* die Stilattribute des untergeordneten Fensters in WS_VISIBLE, WS_CHILD, WS_CLIPSIBLINGS, WS_CLIPCHILDREN, WS_SYSMENU, WS_CAPTION, WS_THICKFRAME, WS_MINIMIZEBOX und WS_MAXIMIZEBOX. Gibt dann den Rückgabewert des Aufrufs von *DefaultProcessing* zurück.

Siehe auch: *TWindow::DefaultProcessing*, Struktur *MDICREATE*

EvMDIDestroy

```
void EvMDIDestroy(HWND hWnd);
```

Fängt die WM_MDIDESTROY-Botschaft ab.

GetClassName

```
char far *GetClassName();
```

Gibt den Namen der Windows-Registrierungsklasse von *TMDIClient* zurück, »MDICLIENT«.

Antworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_COMMAND(CM_ARRANGEICONS, CmArrangeIcons)	CmArrangeIcons
EV_COMMAND(CM_CASCADECHILDREN, CmCascadeChildren)	CmCascadeChildren
EV_COMMAND(CM_CLOSECHILDREN, CmCloseChildren)	CmCloseChildren
EV_COMMAND(CM_CREATECHILD, CmCreateChild)	CmCreateChild
EV_COMMAND(CM_TILECHILDREN, CmTileChildren)	CmTileChildren
EV_COMMAND(CM_TILECHILDRENHORIZ, CmTileChildrenHoriz)	CmTileChildrenHoriz
EV_COMMAND_ENABLE(CM_TILECHILDREN, CmChildActionEnable)	CmChildActionEnable
EV_COMMAND_ENABLE(CM_CASCADECHILDREN, CmChildActionEnable)	CmChildActionEnable
EV_COMMAND_ENABLE(CM_ARRANGEICONS, CmChildActionEnable)	CmChildActionEnable
EV_COMMAND_ENABLE(CM_CLOSECHILDREN, CmChildActionEnable)	CmChildActionEnable
EV_WM_MDICREATE	EvMDICreate
EV_WM_MDIDESTROY	EvMDIDestroy

Klasse TMDIFrame mdi.h

TMDIFrame repräsentiert Rahmenfenster für Multidokumentschnittstellen (MDI, Multiple Document Interface). Diese überlappenden Fenster sind die Hauptfenster MDI-konformer Anwendungen. *TMDIFrame*-Objekte verwalten selbsttätig die Erzeugung und Initialisierung eines MDI-Client-»Fensters«, das von Windows angefordert und durch ein *TMDIClient*-Objekt repräsentiert wird. *TMDIFrame* setzt standardmäßig den Fensterstil WS_CLIPCHILDREN, so daß das Flimmern, das auftritt, wenn MDI seinen Hintergrund und den seiner untergeordneten Fenster löscht, möglichst gering bleibt. *TMDIFrame* ist eine streamfähige Klasse.

Da *TMDIFrame* von *TFrameWindow* abgeleitet ist, erbt es die Bedienbarkeit über die Tastatur. Daher sind auch alle untergeordneten Fenster des MDI-Rahmens über die Tastatur bedienbar. Es ist jedoch zweckmäßig, diese Steuerungsmöglichkeit nur für diejenigen Fenster zu aktivieren, die sie auch benötigen.

Zum Erzeugen eines OLE-fähigen MDI-Rahmenfensters kann *TOleMDIFrame* verwendet werden, das seine Funktionalität sowohl aus *TMDIFrame* als auch aus *TOleFrame* erbt.

Siehe auch: *TOLEMDIFrame*

Öffentliche Konstruktoren

Konstruktor

```
TMDIFrame(const char far *title, TResId menuResId,
          TMDIClient &clientWnd = *new TMDIClient, TModule* module = 0);
```

Erzeugt ein MDI-Rahmenfenster-Objekt und verwendet dazu den Titel (*title*) und die Ressourcenkennung (*menuResId*). Wenn kein Client-Fenster festgelegt ist (*clientWnd*), wird automatisch eine Instanz von *TMDIClient* erzeugt und als Client-Fenster des Rahmens verwendet. Die angegebene Bibliothekskennung wird, zusammen mit einem Null-Zeiger auf das übergeordnete Fenster, dem Titel, dem Client-Fenster und einem Flag an den *TFrameWindow*-Konstruktor übergeben. Das Flag zeigt an, daß die Größe des Client-Fensters nicht angepaßt wird. Außerdem wird der *TWindow*-Konstruktor aufgerufen; er übergibt den angegebenen Titel, die Bibliotheks-ID und einen Null-Zeiger auf das übergeordnete Fenster. Dann wird die Menüposition des untergeordneten Fensters als das am weitesten links gelegene Menüelement initialisiert. Die angegebene Menü-Ressourcen-ID wird in einem Aufruf von AssignMenu verwendet.

Konstruktor

```
TMDIFrame(HWND hWindow, HWND clientHWnd, TModule* module = 0);
```

Erzeugt ein MDI-Rahmenfenster und verwendet dazu ein bereits erzeugtes Fenster, das nicht von ObjectWindows stammt. Ruft die *TFrameWindow*- und *TWindow*-Konstruktoren auf und übergibt ihnen die Fensterkennung (*hWindow*) und die Bibliothekskennung (*module*). Initialisiert die Menüposition des untergeordneten Fensters als am weitesten links gelegenes Menüelement und erzeugt ein *TMDIClient*-Objekt, das dem angegebenen *clientHWnd* entspricht.

Siehe auch: *TFrameWindow::AssignMenu, TFrameWindow::TFrameWindow, TWindow::TWindow,*
TMDIClient::TMDIClient

Öffentliche Elementfunktionen

FindChildMenu

```
static HMENU FindChildMenu(HMENU newMenu);
```

FindChildMenu durchsucht, von rechts nach links, die untergeordneten Popup-Menüs, die in der Menüressource *newMenu* für ein Untermenü enthalten sind, und die Befehlselemente mit einem der folgenden Bezeichner aufweisen: CM_CASCADECHILDREN, CM_TILECHILDREN oder CM_ARRANGEICONS. Der Rückgabewert von *FindChildMenu* ist das *HMENU* des ersten Untermenüs, das einen dieser Bezeichner enthält. Wird keiner der angeführten Bezeichner gefunden, gibt *FindChildMenu* 0 zurück.

FindChildMenu wird verwendet, um die Position des Menüs zu bestimmen, an das die Liste des untergeordneten MDI-Fensters angehängt wer-

den soll. Dieser Aufruf von *FindChildMenu* wird aus der Klasse *TMDIClient* heraus vorgenommen.

GetClientWindow

```
TMDIClient *GetClientWindow ();
```

Gibt einen Zeiger auf das MDI-Client-Fenster zurück.

Siehe auch: *TFrameWindow::GetClientWindow*

GetCommandTarget

```
virtual HWND GetCommandTarget();
```

Ermittelt die Position des untergeordneten Fensters, das Ziel der Befehle und Befehlsaktivierungs-Botschaften ist, und liefert dieses zurück. Wenn die aktuelle Anwendung nicht den Eingabefokus hat oder wenn der Fokus bei einer Mauspalette der Anwendung liegt, gibt *GetCommandTarget* das zuletzt aktive untergeordnete Fenster zurück. Wird eine andere Form der Befehlsverarbeitung gewünscht, kann die Hauptfensterklasse eines Benutzers diese Funktion überschreiben.

PerformCreate

```
void PerformCreate(int menuOrId);
```

Überschreibt die virtuelle Funktion von *TWindow*. Erzeugt das Oberflächenelement, das mit dem MDI-Rahmenfenster verbunden ist.

SetMenu

```
bool SetMenu (HMENU);
```

Sucht in der neuen Menüleiste nach dem MDI-Untermenü und aktualisiert *ChildMenuPos*, wenn das Menü gefunden wird. Sucht in der neuen Menüleiste nach dem Menü des untergeordneten MDI-Fensters und aktualisiert seine Position (*ChildMenuPos*) mit dem festgelegten Menüindex. Sendet dem Client-Fenster eine WM_MDISETMENU-Botschaft, um das neue Menü zu setzen und ruft *TWindow::DrawMenuBar* auf, das Menü neu zu zeichnen. Liefert **false** zurück, wenn das MDI-Client-Fenster angibt, daß es kein Vorgängermenü gegeben hat; gibt andernfalls **true** zurück.

Siehe auch:: *TWindow::DrawMenuBar*

Geschützte Elementfunktion

DefWindowProc

```
LRESULT DefWindowProc (uint message, WPARAM wParam, LPARAM lParam);
```

Überschreibt *TWindow::DefWindowProc* und ruft die Windows-API-Funktion *::DefFrameProc* auf, die für alle eingehenden Botschaften, die nicht vom MDI-Rahmenfenster verarbeitet werden, eine Standardbearbeitung zur Verfügung stellt.

Siehe auch: *::DefFrameProc, ::DefWindowProc*

Antworttabelleneinträge

Die Antworttabelle von *TMDIFrame* hat keine Einträge.

Aufzählungstyp TMeasurementUnits — layoutco.h

```
enum TMeasurementUnits;
```

TMeasurementUnits wird von der Struktur *TLayoutConstraint* verwendet und zählt die Maßeinheiten (*lmPixels* oder *lmLayoutUnits*), die die Abmessungen des Fensters steuern. Diese können entweder Pixel oder Layout-einheiten sein. Layout-Einheiten erhält man durch Teilung der Schrifthöhe in acht vertikale und acht horizontale Abschnitte.

Siehe auch: Struktur *TLayoutConstraint*

Klasse TMemoryDC — dc.h

TMemoryDC ist eine von *TDC* abgeleitete DC-Klasse und bietet Zugriff auf einen Speicher-Gerätekontext (DC).

Öffentliche Konstruktoren

Konstruktor

```
TMemory DC();
```

Standardkonstruktor für ein Speicher-DC-Objekt.

Siehe auch: *TDC::TDC*

Konstruktor

```
TMemoryDC(const TDC& DC);
```

Erzeugt ein Speicher-DC-Objekt, das zum angegebenen *DC*-Argument kompatibel ist.

Siehe auch: *TDC::TDC*

Öffentliche Elementfunktionen

RestoreBitmap

```
void RestoreBitmap();
```

Stellt das ursprünglich ausgewählte Bitmap-Objekt für diesen Gerätekontext wieder her.

Siehe auch: *TDC::RestoreObjects*

RestoreObjects

```
void RestoreObjects();
```

Stellt die ursprünglich ausgewählten Pinsel-, Stift-, Schrift-, Paletten- und Bitmap-Objekte für diesen Gerätekontext wieder her.

Siehe auch: *TDC::RestoreObjects, TMemoryDC::RestoreBitmap*

SelectObject

```
void SelectObject(const TBrush& brush);
void SelectObject(const TPen& pen);
```

```
void SelectObject(const TFont& font);
void SelectObject(const TPalette& palette, bool
                        forceBackground=false);
void SelectObject(const TBitmap& bitmap);
```

Wählt das angegebene GDI-Objekt und stellt es diesem Gerätekontext zur Verfügung.

Siehe auch: *TDC::SelectObject, TMemoryDC::RestoreObjects, TMemoryDC::RestoreBitmap*

Geschütztes Datenelement

OrgBitmap

```
HBITMAP OrgBitmap;
```

Die ursprünglich für diesen Gerätekontext ausgewählte Bitmap.

Siehe auch: *TMemoryDC::SelectObject, TMemoryDC::RestoreBitmap*

Klasse TMenu menu.h

Die *TMenu*-Klasse kapselt Fenstermenüs ein. Elementfunktionen von *TMenu* können verwendet werden, um Menüobjekte zu konstruieren, zu ändern, zu erzeugen und abzufragen. *TMenu* kann außerdem benutzt werden, um Menüs Bitmap-Objekte hinzuzufügen oder festzulegen, ob ein Menüeintrag markiert oder eine vorhandene Markierung entfernt wird.*TMenu* enthält zwei Fassungen einer Helferfunktion *DeepCopy*, die Kopien von Menüs anfertigen und diese an einer angegebenen Position in die Menüleiste einfügen kann. Zum Erzeugen von Menüobjekten siehe das *ObjectWindows Programmierhandbuch*.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TMenu(TAutoDelete autoDelete = AutoDelete);
```

Erzeugt ein leeres Menü und setzt als Voreinstellung *autoDelete*, so daß das Menü beim Entfernen des Objekts automatisch gelöscht wird.

Konstruktor

```
TMenu(const TMenu& original, TAutoDelete autoDelete = NoAutoDelete);
```

Erzeugt eine vollständige Kopie eines bestehenden Menüs und setzt standardmäßig *autoDelete*, so daß das Menü nicht automatisch gelöscht wird, wenn das Objekt entfernt wird.

Konstruktor

```
TMenu(HWND wnd, TAutoDelete autoDelete = NoAutoDelete);
```

Erzeugt ein Menüobjekt, das das aktuelle Menü des Fensters repräsentiert, und setzt standardmäßig *autoDelete*, so daß das Menü nicht automatisch gelöscht wird, wenn das Objekt entfernt wird.

Konstruktor

```
TMenu (HMENU handle, TAutoDelete autoDelete = NoAutoDelete);
```

Erzeugt ein Menüobjekt aus einem bereits geladenen Menü und setzt standardmäßig *autoDelete*, so daß das Menü beim Dekonstruieren des Objekts nicht automatisch gelöscht wird.

Konstruktor

```
TMenu(const void far* menuTemplate);
```

Erzeugt ein Menüobjekt aus einer Menüvorlage im Speicher. Dieser Konstruktor steht unter Presentation Manager nicht zur Verfügung.

Konstruktor

```
TMENU(HINSTANCE instance, TResId resId);
```

Erzeugt ein Menüobjekt aus einer festgelegten Ressourcen-ID.

Destruktor

```
virtual ~TMenu();
```

Entfernt das Menü.

Siehe auch: *Klasse TResId*

Öffentliche Elementfunktionen

AppendMenu

```
bool AppendMenu(uint flags, uint NewItem, const TBitmap& newBMP);
```

Fügt am Ende des Menüs ein Bitmap-Menüelement an. Eine Beschreibung der Flag-Werte, die die Menüattribute festlegen - z.B. Menüelement ist markiert, Menüelement ist deaktiviert, usw. - finden Sie unter *TMenu::GetMenuState*.

AppendMenu

```
bool AppendMenu(uint flags, uint NewItem = -1,
                    const char far* newItem = 0);
```

Fügt am Ende des Menüs ein Text-Menüelement an. Eine Beschreibung der Flag-Werte, die die Menüattribute festlegen - z.B. Menüelement ist markiert, Menüelement ist eine Bitmap, usw. - finden Sie unter *TMenu::GetMenuState*.

Siehe auch: Klasse *TBitmap, TMenu::GetMenuState*

CheckMenuItem

```
bool CheckMenuItem(uint Item, uint check);
```

Markiert den Menüeintrag oder entfernt die vorhandene Markierung. Durch Kombination von Flags mit dem bitweisen ODER-Operator (|) gibt *checks* sowohl die Position des Elements *item* an (MF_BYCOMMAND, MF_BYPOSITION) als auch, ob es markiert (MF_CHECKED)oder demarkiert (MF_UNCHECKED) werden soll.

CheckValid

```
void CheckValid(uint resId, IDS_MENUFAILURE);
```

Löst eine TXMenu-Exception aus, wenn das Objekt ungültig ist.

Siehe auch: *TMenu::TXMenu*

DeleteMenu

```
bool DeleteMenu(uint Item, uint flags);
```

Entfernt den Menüeintrag (*idItem*) aus dem Menü, löscht ihn in Popup-Menüs. *flags* wird verwendet, um die Position des Menüeintrags entweder anhand seiner relativen Position im Menü zu ermitteln (MF_BYPOSITION) oder durch Referenzierung seines Handles an das Hauptmenü (MF_BYCOMMAND).

Siehe auch: *TMenu::RemoveMenu*

DrawItem

```
virtual void DrawItem(DRAWITEMSTRUCT FAR& drawItem);
```

Antwortet auf eine Botschaft, die von *TWindow* an ein zeichenbares Dialogelement weitergereicht wurde, wenn das Dialogelement gezeichnet werden soll.

Siehe auch: Struktur *DRAWITEMSTRUCT*

EnableMenuItem

```
bool EnableMenuItem(uint Item, uint enable);
```

Aktiviert oder deaktiviert das in *item* festgelegte Menüelement, oder stellt es grau dar. Ein aktivierter Menüeintrag (Standardstatus) kann ausgewählt und verwendet werden. Ein grau dargestellter Menüeintrag wird mit grauer Schrift dargestellt und kann vom Benutzer nicht ausgewählt werden. Deaktivierte Menüeinträge werden nicht angezeigt. Gibt bei Erfolg **true** zurück.

GetHandle

```
virtual HMENU GetHandle();
```

Gibt das Handle auf das Menü zurück.

Siehe auch: *TMenu::IsOk*

GetMenuCheckMarkDimensions

```
static bool GetMenuCheckMarkDimensions(TSize& size);
```

Ermittelt die Größe der Bitmap, die zur Darstellung des Standard-Markierungszeichens an markierten Menüeinträgen verwendet wird.

Siehe auch: *TMenu::SetMenuItemBitmaps, Klasse TSize*

GetMenuItemCount

```
uint GetMenuItemCount() const;
```

Gibt die Anzahl der Einträge in einem Hauptmenü oder einem Popup-Menü zurück.

GetMenuItemID

```
uint GetMenuItemID(int posItem) const;
```

Gibt die ID des Menüeintrags an der durch *posItem* festgelegten Position zurück.

GetMenuState

```
uint GetMenuState(uint item, uint flags) const;
```

Gibt die Werte der Flags für ein durch *item* festgelegtes Menüelement zurück. *flags* gibt an, wie *item* interpretiert wird, und ist einer der folgenden Werte:

Flag	Beschreibung
MFBYCOMMAND	*item* wird als Menübefehls-ID interpretiert. Standardeinstellung, wenn weder MF_BYCOMMAND noch MF_BYPOSITION angegeben ist.
MF_BYPOSITION	*item* wird als nullbasierte relative Position des Menüelements im Menü aufgefaßt.

Wird *item* gefunden, und ist es ein Popup-Menü, enthält das niedere Byte des Rückgabewertes die mit *item* verbundenen Flags, das obere Byte die Anzahl der Elemente im Popup-Menü. Ist *item* kein Popup-Menü, bezeichnet der Rückgabewert eine Kombination dieser Flags:

Flag	Beschreibung
MF_BITMAP	Menüelement ist eine Bitmap.
MF_CHECKED	Menüelement ist markiert (nur Popup-Menüs).
MF_DISABLED	Menüelement ist deaktiviert.
MF_ENABLED	Menüelement ist aktiviert. Hinweis: Der Wert dieser Konstante ist 0.
MF_GRAYED	Menüelement ist deaktiviert und grau dargestellt.
MF_MENUBARBREAK	Gleiche Funktion wie MF_MENUBREAK, jedoch werden Spalten in Popup-Menüs durch eine vertikale Linie getrennt.
MF_MENUBREAK	Statische Menüelemente werden in einer neuen Zeile plaziert, Popup-Menüs in einer neuen Spalte, ohne Trennlinie.
MF_SEPARATOR	Es wird eine horizontale Trennlinie gezeichnet, die sich nicht aktivieren, markieren, hervorheben oder grau darstellen läßt. *item* und *flags* werden ignoriert.
MF_UNCHECKED	Die Markierung eines Menüelements wird entfernt (Voreinstellung). Hinweis: Der Wert dieser Konstante ist 0. Liefert -1 zurück, wenn das Element nicht existiert.

GetMenuString

```
uint GetMenuString(uint item, char* str, int count, uint flags)
                                    const;
```

Gibt das Label (*str*) des Menüelements (*idItem*) zurück.

GetSubMenu

```
HMENU GetSubMenu(int posItem) const;
```

Gibt das Handle des durch *posItem* festgelegten Menüs zurück.

InsertMenu

```
bool InsertMenu(uint Item, uint flags, uint NewItem,
                          const TBitmap& newBmp);
```

Fügt ein Bitmap-Menüelement nach dem in *item* festgelegten Menüelement ein. Der Parameter *flags* enthält entweder den Wert MF_BYCOMMAND oder MF_BYPOSITION, der angibt, wie der Parameter *item* zu interpretieren ist. MF_BYCOMMAND kennzeichnet *item* als eine Befehls-ID; bei MF_BYPOSITION hält das Element eine relative Position innerhalb des Menüs.

InsertMenu

```
bool InsertMenu(uint item, uint flags, uint NewItem = -1,
                const char far* newItem = 0);
```

Fügt nach dem in *item* festgelegten Menüelement ein Text-Menüelement oder ein Popup-Menü in das Menü ein. Der Parameter *flags* enthält entweder den Wert MF_BYCOMMAND oder MF_BYPOSITION, der angibt, wie der Parameter *item* zu interpretieren ist. MF_BYCOMMAND kennzeichnet *item* als eine Befehls-ID; bei MF_BYPOSITION hält das Element eine relative Position innerhalb des Menüs.

Siehe auch: *TMenu::GetMenuState*

IsOK

```
bool IsOK() const;
```

Liefert **true** zurück, wenn das Menü ein gültiges Handle hat.

MeasureItem

```
virutal void MeasureItem(MEASUREITEMSTRUCT far& measureItem);
```

Wird von Dialogelementen verwendet, die durch ein übergeordnetes Fenster gezeichnet werden, um die Dimensionen des angegebenen Elementes zu speichern.

Siehe auch: Struktur MEASUREITEMSTRUCT

ModifyMenu

```
bool ModifyMenu(uint idItem, uint flags, uint idNewItem,
                const TBitmap& newBmp);
```

Ersetzt ein bestehendes Menüelement durch eine Bitmap. Der Parameter *flags* enthält entweder den Wert MF_BYCOMMAND oder MF_BYPOSITION, der angibt, wie der Parameter *item* zu interpretieren ist. MF_BYCOMMAND kennzeichnet *item* als eine Befehls-ID; bei MF_BYPOSITION hält das Element eine relative Position innerhalb des Menüs.

ModifyMenu

```
bool ModifyMenu(uint idItem, uint flags, uint idNewItem = -1,
                const char far* newItem = 0);
```

Ersetzt das in *idItem* festgelegte, bestehende Menüelement durch *idNewItem*. Der Parameter *flags* enthält entweder den Wert MF_BYCOMMAND oder MF_BYPOSITION, der angibt, wie der Parameter *item* zu interpretieren ist. MF_BYCOMMAND kennzeichnet *item* als eine Befehls-ID; bei MF_BYPOSITION hält das Element eine relative Position innerhalb des Menüs.

Siehe auch: *TMenu::GetMenuState*

operator HMENU

```
operator HMENU();
```

Gibt das Handle des Menüs zurück.

Siehe auch: *TMenu::uint*

operator =

```
TMenu& operator =(const TMenu& original);
```

Liefert das Objekt *TMenu* zurück.

Siehe auch: TMenu::operator uint

operator uint

```
operator uint();
```

Gibt das Handle des Menüs zurück. Diese Funktion bietet Kompatibilität zu einigen Windows-Funktionen, die einen Menüparameter uint erfordern.

Siehe auch: *TMenu::HMenu*

RemoveMenu

```
bool RemoveMenu(uint idItem, uint flags);
```

Entfernt das Menüelement aus dem Menü, löscht es aber nicht, wenn es sich um ein Untermenü handelt.

Siehe auch: *TMenu::DeleteMenu*

SetMenuItemBitmaps

```
inline bool SetMenuItemBitmaps(uint idItem, uint flags,
                               const TBitmap* bmpUnchecked=0,
                               const TBitmap* bmpChecked=0);
                               static bool
GetMenuCheckmarkDimensions(TSize& size);
```

Legt die anzuzeigende Bitmap fest, wenn ein Menüelement markiert bzw. seine Markierung aufgehoben wird. *idItem* zeigt den Menüeintrag an, an den die Bitmap gebunden wird. *flags* zeigt an, wie der *size*-Parameter interpretiert wird (ob durch MF_BYPOSITION oder durch MF_BYCOMMAND). *GetMenuCheckmarkDimensions* ermittelt die Größe der Bitmap.

Siehe auch: *TMenu::GetMenuCheckmarkDimensions, TBitmap*

Geschützte Datenelemente

Handle

```
HMENU Handle;
```

Enthält das Handle des Menüs.

ShouldDelete

```
bool ShouldDelete;
```

ShouldDelete ist auf **true** gesetzt, wenn der Destruktor das Handle des Menüs löschen muß.

Siehe auch: *TMenu::DeleteMenu, TMenu::RemoveMenu*

Geschützte Elementfunktionen

DeepCopy

```
static void DeepCopy(TMenu& dest, const TMenu& source, int offset = 0,
int count = -1);
```

Erstellt eine »tiefe« Kopie des Menüs (also eine wirkliche Kopie des Menüs, nicht nur seiner Zeiger oder Handles). Diese Form von *DeepCopy* kopiert eine Anzahl *count* von Popup-Menüs oder Menüelementen aus *src*, beginnend bei *offset*, und hängt die Elemente oder Menüs an das Ziel-

menü an. Wird *count* als -1 übergeben, werden alle Elemente des Quellmenüs kopiert.

DeepCopy

```
static void DeepCopy(TMenu& dest, int dstOff, const TMenu& src, int
stcOff = 0, int count = -1);
```

Erstellt eine »tiefe« Kopie des Menüs (also eine wirkliche Kopie des Menüs, nicht nur seiner Zeiger oder Handles). Diese Form von *DeepCopy* kopiert eine Anzahl *count* von Popup-Menüs oder Menüelementen aus *src*, beginnend bei *offset*, und fügt die Elemente oder Menüs an der Position dstOffset in das Zielmenü ein. Wird *count* als -1 übergeben, werden alle Elemente des Quellmenüs kopiert.

Klasse TMenuDescr framewin.h

TMenuDescr beschreibt die Menüleiste und ihre funktionalen Gruppen. *TMenuDescr* bietet Ihnen eine einfache Möglichkeit, Menüs in Ihrer Menüleiste zu gruppieren und einer existierenden Menüleiste neue Gruppen hinzuzufügen.*TMenuDescr* benutzt eine Ressource-ID zur Identifikation der Menüressource und ein Array von Zählerwerten, um die Anzahl von Menüs in jeder Gruppe der Menüleiste anzuzeigen.

Der Aufzählungstyp *TGroup* zählt die sechs grundlegenden Funktionsgruppen einer Menüleiste auf: Datei, Bearbeiten, Container, Objekt, Fenster und Hilfe. Praktisch initialisieren die Konstruktoren von *TMenuDescr* lediglich die Elemente nach Maßgabe der übergebenen Argumente: das eigentliche Kombinieren der Menü- gruppen wird von der Funktion *MergeMenu* von *TFrameWindow* durchgeführt.

Eine Methode zum Erzeugen eines Menüs besteht im Aufrufen des Konstruktors *TMenuDescr* und im Übergeben des Anzahl der Gruppenzähler für jede Menüauswahl.

Wenn beispielsweise das ursprüngliche Menü wie folgt aussah:

Datei Bearbeiten Suchen Ansicht Seite Absatz Wort Fenster Hilfe

könnten die folgenden Gruppenzähler verwendet werden:

Gruppe	Zähler	Menü
Dateigruppe	1	Datei
Gruppe Bearbeiten	2	Bearbeiten Suchen
Container-Gruppe	1	Ansicht
ObjektGruppe	3	Seite Absatz Wort
Fenstergruppe	1	Fenster
Hilfegruppe	1	Hilfe

Dann wird der Konstruktor wie folgt aufgerufen:

```
TMenuDescr(IDM_MYMENU, 1, 2, 1, 3, 1, 1)
```

Das oben dargestellte Menü läßt sich durch Verschmelzen zweier Menüs erzeugen. Beispielsweise ist das Rahmenmenü der Anwendung wie folgt aufgebaut:

Datei Ansicht Fenster Hilfe

```
TMenuDescr(IDM_FRAME, 1, 0, 1, 0, 1, 1)
```

und die Menüleiste der untergeordneten Textverarbeitung so:

Bearbeiten Suchen Seite Absatz Wort Hilfe

```
TMenuDescr(IDM_WPROC, 0, 2, 0, 3, 0, 1)
```

Wenn keine untergeordnete Anwendung aktiv ist, wird nur das Rahmenmenü aktiv sein. Wird das untergeordnete Fenster der Textverarbeitung aktiviert, wird die untergeordnete Menüleiste mit dem Rahmenmenü kombiniert. Jede Gruppe der untergeordneten Menüleiste, die 0 gesetzt ist, läßt die übergeordnete Gruppe unverändert. In diesem Beispiel wechseln die Gruppen ab, mit Ausnahme der letzten: Die Hilfegruppe der untergeordneten Anwendung ersetzt die des Rahmenmenüs.

Üblicherweise gehören die »geraden« Gruppen (Datei, Container, Fenster) zum Rahmenmenü oder Container, die »ungeraden« (Bearbeiten, Objekt, Hilfe) zur untergeordneten oder eingeschlossenen Gruppe.

Bei Verwendung von -1 als Gruppenzähler wird die übergeordnete Gruppe ersatzlos entfernt. Beispielsweise könnte eine weitere untergeordnete Menüleiste, etwa für einen Taschenrechner, folgendermaßen in die Anwendung einbezogen werden:

Bearbeiten Basis Hilfe

```
TMenuDescr(IDM_WCALC, 0, 1, -1, 1, 0, 1)
```

Es ergibt sich das folgende zusammengesetzte Menü (ohne die Auswahloption Ansicht, die durch die Angabe -1 entfernt wurde):

Datei Bearbeiten Basis Fenster Hilfe

Ein Zeichenfenster könnte folgendermaßen hinzugefügt werden:

Bearbeiten Bitmap Pixel Hilfe

```
TMenuDescr(IDM_WPAINT, 0, 1, 0, 2, 0, 1)
```

Es ergibt sich dann das folgende kombinierte Menü:

Datei Bearbeiten Ansicht Bitmap Pixel Fenster Hilfe

Die einfachste Methode, einer Menüleiste Gruppen hinzuzufügen, besteht im Definieren der Menügruppen und Hinzufügen von Separatoren

in einer Ressourcendatei. Fügen Sie zwischen jeder der Gruppen den Ausdruck MENUITEM SEPARATOR ein, und einen zusätzlichen Separator, wenn eine der Menügruppen nicht anwesend ist. Beispielsweise definiert die Ressourcendatei für Schritt 14 des ObjectWindows-Tutorials die folgenden Menügruppen und Separatoren:

```
IDM_MDICMNDS MENU
{
// Datei-Menü grau anzeigen
 MENUITEM "File",   0,GRAYED  ;Platzhalter für Dateimenü des
Dokumentverwalters
 MENUITEM SEPARATOR
 MENUITEM "Edit",   CM_NOEDIT ;Platzhalter für das Menü Bearbeiten aus
der View
 MENUITEM SEPARATOR
 MENUITEM SEPARATOR
 MENUITEM SEPARATOR
 POPUP "&Window"
{
// Optionen in der Menügruppe Fenster
  MENUITEM  "&Cascade",       CM_CASCADECHILDREN
  MENUITEM  "&Tile",          CM_TILECHILDREN
  MENUITEM  "Arrange &Icons", CM_ARRANGEICONS
  MENUITEM  "C&lose All",     CM_CLOSECHILDREN
  MENUITEM  "Add &View",      CM_VIEWCREATE
 }
 MENUITEM SEPARATOR
 POPUP "&Help"
 {
  MENUITEM "&About",   CM_ABOUT
 }
}
```

(Sie können sich die Separatoren anzeigen lassen, indem Sie STEP14.RC in den Resource Workshop laden und die View als Popup-Option im View-Menü deaktivieren.)

Diese Ressourcendatei definiert eine Gruppe Bearbeiten, eine Gruppe Datei, eine Gruppe Fenster und eine Hilfe-Gruppe, jedoch keine Einträge für die Gruppen Container und Objekt.

STEP14.CPP verwendet dann diese Befehle aus der Ressourcendatei, um das Hauptfenster und sein Menü aufzubauen. Dabei wir der Funktion *SetMenuDescr* IDM_MDICMNDS als Parameter in dieser Weise übergeben

```
SetMainWindow(frame);
GetMainWindow()->SetMenuDescr(TMenuDescr(IDM_MDICMNDS));
```

und die folgenden Menügruppen erzeugt:

```
Datei  Bearbeiten      Fenster  Hilfe
```

Die Funktionen von *TMenuDescr* erlauben den Aufbau von kombinierten Menüs im Stil von OLE2. Das heißt, die Menüs eines Container-Dokuments (des MDI-Rahmenfensters) lassen sich mit denen eines eingebetteten Objekts (des untergeordneten MDI-Fensters) kombinieren. Wird das eingebettete Objekt durch Doppelklicken mit der Maus an Ort und Stelle aktiviert, wird das Menü des untergeordneten Fensters mit dem des Rahmenfensters kombiniert.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TMenuDescr();
```

Standardkonstruktor für ein *TMenuDescr*-Objekt. Es werden keine Menüressourcen oder Gruppen angegeben. Es wird eine leere Menüleiste konstruiert.

Konstruktor

```
TMenuDescr(TResId id, int fg, int eg, int cg, int og, int wg, int hg,
TModule* module = ::Module);
```

Konstruiert einen Menüdeskriptor aus den mit den Parametern *id* und *module* bezeichneten Ressourcen. Positioniert die Popup-Menüs in Gruppen, nach Maßgabe der Werte der Parameter *fg, eg, cg, of, wg* und *hg*. Die Parameter *fg, eg, cg, of, wg* und *hg* repräsentieren die funktionalen Gruppen, die im Aufzählungstyp *TGroup* identifiziert sind. Ruft die Funktion *ExtractGroups* auf, um auf der Grundlage der Separator-Elemente die Gruppenzähler in der Menüleiste zu extrahieren.

Konstruktor

```
TMenuDescr(const TMenuDescr& original);
```

Kopiert das Menüdeskriptor-Objekt, das im Parameter *original* angegeben ist.

Konstruktor

```
TMenuDescr(TResId id, TModule* module = ::Module);
```

Erzeugt einen Menüdeskriptor aus der im Parameter *id* angegebenen Menüresource. Ruft die Funktion *ExtractGroups* auf, um auf der Grundlage der Separator-Elemente die Gruppenzähler in der Menüleiste zu extrahieren.

Konstruktor

```
TMenuDescr(HMENU hMenu, int fg, int eg, int cg, int of, int wg, int hg,
TModule* module = ::Module);
```

Konstruiert einen Menüdeskriptor aus dem Menü-Handle, das im Parameter *hMenu* angegeben ist. Der Menüdeskriptor kann null oder mehr Popup-Menüs in mehr als einer funktionalen Gruppe haben. Die Parameter *fg, eg, cg, of, wg* und *hg* repräsentieren die funktionalen Gruppen, die im Aufzählungstyp *TGroup* identifiziert sind. Ruft die Funktion *ExtractGroups* auf, um auf der Grundlage der Separator-Elemente die Gruppenzähler in der Menüleiste zu extrahieren, oder verwendet die angegebenen

Gruppenzähler-Parameter, wenn die Menüleiste keine Separator-Elemente enthält.

Destruktor

```
~TMenuDescr();
```

Entfernt das TMenuDescr-Objekt.

Typdefinitionen

TGroup enum

```
enum TGroup{FileGroup, EditGroup, ContainerGroup, ObjectGroup,
WindowGroup, HelpGroup, NumGroups};
```

Der Aufzählungstyp *TGroup* wird von *TMenuDescr* verwendet und beschreibt die folgenden Konstanten, die den Index des Eintrags im *GroupCount*-Array definieren.

Konstante	Bedeutung
FileGroup	Index des Gruppenzählers der Dateigruppe
EditGroup	Index des Gruppenzählers der Menügruppe Bearbeiten
ContainerGroup	Index des Gruppenzählers der Menügruppe Container
ObjectGroup	Index des Gruppenzählers der Menügruppe Objekt
WindowGroup	Index des Gruppenzählers der Menügruppe Fenster
HelpGroup	Index des Gruppenzählers der Menügruppe Hilfe
NumGroups	Gesamtzahl der Gruppen

Siehe auch: *TMenuDescr::GroupCount*

Öffentliche Elementfunktionen

GetHandle

```
HMENU GetHandle() const;
```

Holt das Menü-Handle; unter Umständen kann irgendein verschobener Menüerwerb dadurch ausgelöst werden.

GetModule

```
TModule* GetModule() const;
```

Liefert einen Zeiger auf das Modulobjekt zurück.

SetModule

```
void SetModule(TModule* module);
```

Setzt das Standard-Modulobjekt für diesen Menüdeskriptor.

GetId

```
TResId GetId() const;
```

Holt die Menüressourcenkennung, die zum Konstruieren des Menüdeskriptors verwendet wird.

GetGroupCount

```
int GetGroupCount(int group) const;
```

Holt die Anzahl der Menüs in einer angegebenen Gruppe innerhalb der Menüleiste. Die Maximalzahl der funktionalen Gruppen liegt bei sechs,

wie im Aufzählungstyp *TGroup* festgelegt. Diese Gruppen sind *FileGroup*, *EditGroup*, *ContainerGroup*, *ObjectGroup*, *WindowGroup* und *HelpGroup*.

Siehe auch: Aufzählungstyp TGroup

ClearServerGroupCount `void ClearServerGroupCount();`

Löscht die ungeraden Gruppen (also 1, 3 und 5, bzw. Gruppe Bearbeiten, Objektgruppe und Hilfe-Gruppe) in der Menüleiste einer Server-Anwendung.

Siehe auch: Aufzählungstyp TGroup, TOcMenuDescr

ClearContainerGroupCount `void ClearContainerGroupCount();`

Löscht die geraden Gruppen (also 0, 2 und 4, bzw. Dateigruppe, Container-Gruppe und Fenstergruppe) in der Menüleiste einer Container-Anwendung.

Siehe auch: Aufzählungstyp TGroup, TOcMenuDescr

Merge

```
bool Merge(const TMenuDescr& sourceMenuDescr);
```

Blendet die funktionalen Gruppen eines anderen Menüdeskriptors in diesen Menüdeskriptor ein.

Merge

```
bool Merge(const TMenuDescr& sourceMenuDescr, TMenu& destMenu);
```

Kombiniert die funktionalen Gruppen dieses Menüdeskriptors mit denen eines anderen, und setzt beides in ein leeres Menü ein.

Siehe auch: *TMenuDesc::TGroup enum*

Geschützte Datenelemente

Id

```
TResId Id;
```

Ressourcen-ID für das Menü. Die Ressourcenkennung wird in den Konstruktoren übergeben, um die Menüressource zu identifizieren.

GroupCount

```
int GroupCount[NumGroups];
```

Ein Array von Werten, das die Anzahl der Popup-Menüs innerhalb jeder Gruppe der Menüleiste anzeigt.

Siehe auch: Aufzählungstyp *TGroup*

Module

```
TModule* Module;
```

Zeigt auf das Objekt *TModule*, dem der Deskriptor *TMenuDescr* gehört.

Geschützte Elementfunktionen

ExtractGroups

```
bool ExtractGroups();
```

Extrahiert die Gruppenzähler aus der geladenen Menüleiste, indem es die Anzahl der Menüelemente zwischen den Separatorelementen zählt. Die Separatoren werden entfernt, nachdem die Gruppenzähler extrahiert wurden.

Siehe auch: *TMenu, TOcMenuDescr*

Klasse TMenuItemEnabler — framewin.h

TMenuItemEnabler ist von *TCommandEnabler* abgeleitet und stellt einen Befehlsaktivierer für Menüelemente dar. Die Funktionen dieser Klasse modifizieren Text, Markierungszustand und Aussehen von Menüelementen.

Öffentlicher Konstruktor

Konstruktor

```
TMenuItemEnabler(HMENU hMenu, uint id, HWND hWndReceiver, int position);
```

Konstruiert ein Objekt *TMenuItemEnabler* mit der angegebenen Befehlskennung für das angegebene Menüelement, den Botschaftsempfänger (*hWndReceiver*) und die Menüposition.

Geschützte Datenelemente

HMENU

```
HMENU HMenu;
```

Das Menü, das das zu aktivierende oder deaktivierende Element enthält.

Position

```
int Position;
```

Die Position des Menüelements.

Öffentliche Elementfunktionen

Enable

```
void Enable(bool enable);
```

Überschreibt *TCommandEnable::Enable*. Aktiviert oder deaktiviert die Menüoptionen, die die Darstellung des entsprechenden Menüelements steuern.

GetMenu

```
HMENU GetMenu();
```

Liefert das Menü zurück, das das zu aktivierende oder deaktivierende Menüelement enhält.

GetPosition

```
int GetPosition();
```

Gibt die Position des Menüelements zurück.

SetText

```
void SetText(LPCSTR);
```

Überschreibt *TCommandEnable::SetText*. Ändert den Text des entsprechenden Menüelements.

SetCheck

```
void SetCheck(int state);
```

Überschreibt *TCommandEnable::SetCheck*. Markiert oder demarkiert das entsprechende Menüelement. Der Parameter *Status* spiegelt den Zustand des Menüelements - markiert, nicht markiert oder unbestimmt - wieder.

Klasse TMessageBar — messageb.h

TMessageBar ist aus *TGadgetWindow* abgeleitet und implementiert eine Meldungszeile mit einem randlosen Textgadget von Fensterbreite. Die Meldungszeile ist normalerweise an der Unterkante des Fensters positioniert und verwendet die für Gadgets voreingestellte Schrift; sie wird nach oben durch eine hervorgehobene Linie abgeschlossen.

Öffentlicher Konstruktor

Konstruktor

```
TMessageBar(TWindow* parent = 0, TFont* font = new TGadgetWindowFont,
            TModule* module = 0);
```

Konstruiert ein *TMessageBar*-Objekt mit der Schriftart des Gadget-Fensters. Setzt IDW_STATUSBAR, *HighlightLine* auf **true** und das *TTextGadget*-Element *WideAsPossible* auf **true**, wodurch das Textgadget die ganze Fensterbreite einnimmt.

Siehe auch: *TGadgetWindowFont::TGadgetWindowFont*

Öffentliche Elementfunktionen

SetHintText

```
virtual void SetHintText(const char* text);
```

Setzt oder löscht den Menühinweistext für die Meldungszeile. Hinweistext wird über allen anderen Gadgets angezeigt und wird für die Menühilfe und die Hilfe zu den Schaltern der Dialogelementleiste verwendet.

SetText

```
void SetText(const char* text);
```

Leitet die Meldung in der Meldungszeile zur Formatierung an das Text-Gadget weiter.

Siehe auch: *TTextGadget::SetText*

Geschützte Datenelemente

HighlightLine

```
bool HighlightLine;
```

Ist **true**, wenn eine hervorgehobene Linie gezeichnet wird.

HintText

```
char* HintText;
```

Speichert den aktuell angezeigten Kommandohinweistext, sofern einer vorliegt.

Geschützte Elementfunktionen

GetDesiredSize

```
void GetDesiredSize(TSize& rect);
```

Ruft *GetDesiredSize* von *TGadgetWindow* auf, um die Größe der Meldungszeile zu ermitteln. Paßt dann ihre Größe an, wenn eine hervorgehobene Linie gezeichnet wird.

Siehe auch: *TGadgetWindow::GetDesiredSize*

GetInnerRect

```
void GetInnerRect(TRect& rect);
```

GetInnerRect berechnet das Rechteck innerhalb der Rahmen und Ränder der Meldungszeile.

Siehe auch: *TGadgetWindow::GetInnerRect*

PaintGadgets

```
void PaintGadgets(TDC& dc, bool erase, TRect& rect);
```

Justiert die Meldungszeile und zeichnet eine hervorgehobene Linie. Zeichnet anschließend den Hinweistext, wenn einer gesetzt wurde, oder ruft *TGadgetWindow::PaintGadgets* auf, um alle Gadgets neu zu zeichnen.

Siehe auch: *TGadgetWindow::PaintGadgets*

Klasse TMetaFileDC — dc.h

TMetaFileDC ist aus *TDC* abgeleitet und bietet Zugriff auf einen Gerätekontext mit einem zum Zeichnen ausgewählten Metafile.

Konstruktoren und Destruktor

Konstruktor

```
TMetaFileDC(const char far* filename = 0);
```

Standardkonstruktor für *TMetaFileDC*-Objekte.

Siehe auch: *TDC::TDC*

Konstruktor

```
TMetaFileDC(const char far* fileName = 0);
```

Erzeugt ein *TMetaFileDC*-Objekt und verwendet dazu die Daten in der angegebenen Datei.

Siehe auch: *TDC::TDC*

Destruktor

```
~TMetaFileDC();
```

Entfernt dieses Objekt.

Öffentliche Elementfunktion

Close

```
inline HMETAFILE Close();
```

Schließt dieses Metafile-Gerätekontextobjekt. Setzt das Datenelement *Handle* auf 0 und gibt einen Zeiger auf ein neues *TMetaFilePict*-Objekt zurück.

Siehe auch: *::CloseMetaFile, TMetaFilePict, ::DeleteMetaFile*

Klasse TMetaFilePict — metafile.h

TMetaFilePict ist eine Hilfsklasse, die zusammen mit *TMetaFileDC* verwendet wird und Metadatei-Operationen vereinfacht, wie etwa das Abspielen in einen Gerätekontext oder das Speichern von Daten in der Zwischenablage. *TMetafilePict* verwaltet die Umwandlung zwischen einer Metadatei und einem Metafilepict-Objekt selbsttätig.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TMetaFilePict(HMETAFILE handle, TAutoDelete autoDelete);
```

Erzeugt ein *TMetaFilePict*-Objekt; *Handle* ist auf das angegebene *handle*-Argument gesetzt.

Konstruktor

```
TMetaFilePict(const TClipboard& clipboard);
```

Erzeugt ein *TMetaFilePict*-Objekt aus dem Inhalt der angegebenen Zwischenablage.

Konstruktor

```
TMetaFilePict(const char* filename);
```

Erzeugt ein *TMetaFilePict*-Objekt für die Metadatei, die in der angegebenen Datei gespeichert ist.

Konstruktor

```
TMetaFilePict(uint size, const void far* data);
```

Erzeugt ein *TMetaFilePict*-Objekt für die in *data* bestimmte Metadatei, die im Speicher vorhanden ist. Der *data*-Puffer muß eine Metadatei mit der in *size* angegebenen Länge in Bytes enthalten.

Konstruktor

```
TMetaFilePict(HGLOBAL data);
```

Erzeugt ein *TMetaFilePict*-Objekt für die im Speicher vorhandene und durch *data* bezeichnete Metadatei. Der globale Speicherblock *data* muß eine Metadatei enthalten.

Konstruktor

```
TMetaFilePict(const TMetaFilePict& orig, const char far* filename = 0);
```

Kopiert die Metadatei (*orig*) in die bezeichnete Datei. Wenn *filename* 0 ist (Voreinstellung), wird das Metadatei in eine im Speicher vorhandene Metadatei kopiert.

Destruktor

```
~TMetaFilePict();
```

Entfernt dieses Objekt.

Siehe auch: *TClipboard*

Öffentliche Elementfunktionen

CalcPlaySize

```
TSize CalcPlaySize(TDC& dc, const TSize& defSize) const;
```

Berechnet die Größe dieser Metadatei, wenn sie auf einen gegebenen DC gespielt wird.

Siehe auch: *TDC, TSize*

GetMetaFileBits

```
HANDLE GetMetaFileBits();
```

Gibt das Handle eines globalen Speicherblocks, der diese Metadatei als Bitsammlung enthält, zurück. Der Speicherblock kann benutzt werden, um die Größe der Metadatei zu bestimmen oder um sie als Datei abzuspeichern.

GetMetaFileBitsEx

```
uint32 GetMetaFileBitsEx(uint size, void* data);
```

Nur für 32-Bit. Stellt den Inhalt der zu diesem Objekt gehörenden Metadatei fest und kopiert beides in den *data*-Puffer. Wenn *data* ungleich 0 und der Aufruf erfolgreich ist, wird die tatsächlich kopierte Anzahl Bytes zurückgegeben. Ist *data* 0, gibt ein erfolgreicher Aufruf die vom Puffer benötigte Anzahl Bytes zurück. Ein Rückgabewert von 0 zeigt immer einen Fehler an.

Height

```
int Height() const;
```

Ermittelt die Höhe dieser Metadatei.

operator HMETAFILE()

```
operator HMETAFILE() const;
```

Typumwandlungs-Operator.

operator <<

```
TClipboard& operator <<(TClipboard& clipboard, TMetaFilePict& mfp);
```

Plaziert das *TMetaFilePict*-Objekt in der Zwischenablage. Gibt eine Referenz auf die Zwischenablage zurück, wodurch das übliche Aneinanderketten von **<<**-Operationen möglich ist.

MappingMode

```
unsigned MappingMode();
```

Ermittelt den Abbildungsmodus für diese Metadatei.

PlayOnto

```
bool PlayOnto(TDC& dc, const TSize& defSize) const;
```

Spielt eine Metadatei in einen DC.

Siehe auch: TDC, TSize

SetMappingMode

```
void SetMappingMode(unsigned mm);
```

Setzt den Abbildungsmodus für diese Metadatei.

SetSize

```
void SetSize(const TSize& size);
```

Setzt die Größe dieser Metadatei.

Size

```
TSize Size() const;
```

Ermittelt die Größe dieser Metadatei.

ToClipboard

```
void ToClipboard(TClipboard& clipboard, unsigned mapMode = MM_ANISOTROPIC,
                 const TSize& extent=TSize(0,0));
```

Legt diese Metadatei in der Zwischenablage ab.

Siehe auch: *TClipboard, TSize*

Width

```
inline int Width() const;
```

Ermittelt die Breite dieser Metadatei.

Geschützte Datenelemente

Extent

```
TSize Extent;
```

Enthält die Ausdehnung (d.h. die Größe) der Metadatei.

Mm

```
int Mm;
```

Speichert den Abbildungsmodus für die Metadatei.

Klasse TModule — module.h-

Die dynamischen Linkbibliotheken (DLL, dynamic-link libraries) von ObjectWindows konstruieren eine Instanz von *TModule*, die als objektorientierter Platzhalter für das Bibliotheksmodul fungiert. *TModule* definiert Verhalten, das den Bibliotheks- und Anwendungsmodulen gemeinsam ist. ObjectWindows-Anwendungen konstruieren eine Instanz von *TApplication*, die aus *TModule* abgeleitet ist. Die Konstruktoren von *TModule* verwalten das Laden und Freigeben externer DLLs; die Elementfunktionen bieten Unterstützung für die Standard-Fehlerbehandlung.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TModule(const char far* name, HINSTANCE Instance, const char far* cmdLine);
```

Konstruiert ein *TModule*-Objekt für eine ObjektWindows-DLL oder ein ObjectWindows-Programm aus *LibMain* oder *WinMain* heraus. Initialisiert *hInstance* und *cmdLine* durch Aufruf von *InitModule*.

Konstruktor

```
TModule(const char far* name, HINSTANCE hInstance);
```

Konstruiert ein *TModule*-Objekt, das als Alias für eine bereits geladene DLL oder ein bereits geladenes Programm mit verfügbarer *HInstance* fungiert. Wenn das *TModule* entfernt wird, wird die Instanz nicht automatisch freigegeben. *name* kann 0 sein, die Angabe ist optional.

Konstruktor

```
TModule(const char far* name, bool shouldLoad = true);
```

Konstruiert ein *TModule*-Objekt, das als Alias für eine DLL verwendet wird. *TModule* wird die DLL automatisch laden und freigeben, wenn *shouldLoad* **true** ist. Ist *shouldLoad* **false**, muß die *HInstance* später durch *InitModule* gesetzt werden.

Destruktor

```
virtual ~TModule();
```

Entfernt ein *TModule*-Objekt und löscht *lpCmdLine*.

Öffentliche Datenelemente

lpCmdLine

```
char far* lpCmdLine;
```

lpCmdLine, eine nullterminierte Zeichenkette, zeigt auf eine Kopie der Kommandozeilenargumente, die beim Laden des Moduls übergeben werden. Es ist zu beachten, daß *lpCmdLine* sich von der WIN32-*lpCmdLine*-Funktion unterscheidet, bei der der vollständige Pfadname des Moduls an die Kommandozeilenargumente angehängt wird. Das Datenelement *TModule::lpCmdLine* von ObjectWindows enthält stets nur die Kommandozeilenargumente, gleich ob es unter WIN16 oder WIN32 läuft. Auch die globalen Variablen **_argv*[]** und **_argc** der Laufzeitbibliothek enthalten sowohl für die WIN16- als auch für dieWIN32-API identische Informationen. **_argv[0]** zeigt auf den vollständigen Pfadnamen des Moduls.

Siehe auch: *TApplication*

Module

```
extern TModule *Module;
```

Enthält einen globalen Zeiger auf das aktuelle Modul.

Siehe auch: *TApplication*

Status

```
TStatus Status;
```

Status enthält den Modulstatus und wurde aus Gründen der Abwärtskompatibilität mit ObjectWindows 1.0-Anwendungen aufgenommen. ObjectWindows 2.5 benutzt stattdessen Exceptions zur Fehlerbehandlung. Wird *Status* auf einen Wert ungleich 0 gesetzt, löst das eine *TXCompatibility*-Exception aus.

Siehe auch: *TXCompatibility::MapStatusCodeToString*

Öffentliche Elementfunktionen

AccessResource

```
int AccessResource(HRSRC hRsrc) const;
```

AccessResource wird in 16-Bit-Anwendungen zum Auffinden der festgelegten Ressource eingesetzt. Die Verwendung von *FindResource* ist jedoch vorzuziehen.

Siehe auch: *TModule::FindResource*

AllocResource

```
HGLOBAL AllocResource(HRSRC hRsrc, uint32 size) const;
```

AllocResource wird für 16-Bit-Anwendungen eingesetzt und lädt eine Ressource in den Speicher. Die Verwendung von *LoadResource* ist jedoch vorzuziehen.

Siehe auch: *TModule::LoadResource*

CopyCursor

```
HCURSOR CopyCursor(HCURSOR hCursor) const;
```

CopyCursor wird in 16-Bit-Anwendungen eingesetzt und kopiert den in *hCursor* festgelegten Cursor. Der Rückgabewert ist ein Handle zum kopierten Cursor.

Siehe auch: Klasse *TIcon*

CopyIcon

```
HICON CopyIcon(HICON hIcon) const;
```

CopyIcon kopiert das in *hIcon* festgelegte Symbol. Der Rückgabewert ist ein Handle zum Symbol oder 0 bei Mißerfolg. Die Kopie des Symbols sollte entfernt werden, wenn sie nicht mehr benötigt wird.

Error

```
virtual void Error(int errorCode);
```

Error behandelt Fehler, die durch den in *errorCode* gelieferten Fehlerwert bezeichnet werden. *Error* zeigt den Fehlercode in einem Meldungsfenster an und fragt den Benutzer, ob die Programmausführung fortgesetzt werden soll. Wird sie fortgesetzt, kann sich das Programm vielleicht stabilisieren, vielleicht aber auch nicht. Wird der Programmlauf nicht fortgesetzt, wird das Programm beendet. *Error* kann durch eine andere Fehlerbehandlungsroutine außer Kraft gesetzt werden. Diese Funktion dient nur der Abwärtskompatibilität mit ObjectWindows 1.0. In ObjectWindows 2.5-Anwendungen ist stattdessen die folgende *Error*-Funktion zu verwenden.

Error

```
virtual int Error(xmsg& x, unsigned captionResId, unsigned PromptResId=0);
```

Error wird beim Auftreten schwerwiegender Fehler aufgerufen und nimmt ein *xmsg*-Exception-Objekt, eine Ressourcen-ID für den Titel eines Meldungsfensters auf und eine optionale Ressourcen-ID für eine Benutzerantwort. Standardmäßig ruft *Error HandleGlobalException* mit dem *xmsg*-Objekt und den von den Ressourcen übergebenen Strings auf. Eine Anwendung (aus *TApplication* abzuleiten, das aus *TModule* abgeleitet ist)

kann die Funktion neu implementieren, um ein abweichendes Verhalten festzulegen.

Ein Status-Code ungleich 0 wird zurückgegeben um anzuzeigen, daß eine Fehlerbedingung bekanntzugeben ist; Status 0 zeigt die erfolgreiche Bewältigung der Bedingung an, so daß der Programmlauf fortgesetzt werden kann. ObjectWindows verwendet diesen Satus-Code intern in seiner Meldungsschleife, um dem Programm die Wiederaufnahme zu ermöglichen. Die globale Fehlerbehandlungsroutine (in *except.h* definiert), die den Meldungstext anzeigt, ist

```
int _OWLFUNC HandleGlobalException(xmsg& x, char* caption, char*
                                   canResume);
```

ExecDialog

```
int ExecDialog(TDialog* dialog);
```

Führt ein Dialogfenster aus. Diese Funktion dient nur der Abwärtskompatibilität. Stattdessen sollte *TDialog::Execute* verwendet werden.

FindResource

```
HRSRC FindResource(TResId id, const char far* type) const;
```

Sucht die von *id* und *type* bezeichnete Ressource und gibt bei Erfolg ein Handle auf sie zurück. Wird die Ressource nicht gefunden, ist der Rückgabewert 0. Die *id*- und *type*- Parameter zeigen entweder auf nullterminierte Zeichenketten oder legen einen ganzzahligen Wert fest. *type* kann einer der Standard-Ressourcentypen, die unten definiert sind:

Wert des Typs	Ressource
RT_ACCELERATOR	Tastenkürzel-Tabelle
RT_BITMAP	Bitmap
RT_CURSOR	Cursor
RT_DIALOG	Dialogfenster
RT_FONT	Schriftart
RT_FONTDIR	Schriftartverzeichnis
RT_ICON	Symbol
RT_MENU	Menü
RT_RCDATA	Benutzerdefinierte Ressource
RT_STRING	String

Siehe auch: *TResId*

GetClientHandle

```
HWND GetClientHandle(HWND hWnd);
```

Holt das Handle auf das Client-Fenster. Diese Funktion dient nur der Abwärtskompatibilität mit ObjectWindows 1.0.

GetClassInfo

```
bool GetClassInfo(const char far* name, WNDCLASS far* wndclass) const;
```

GetClassInfo wird insbesondere für die Bildung von Unterklassen verwendet und holt Informationen über die in *wndclass* festgelegte Fensterklasse. *name* zeigt auf eine nullterminierte Zeichenkette, die den Namen der Klas-

se enthält. *wndclass* zeigt auf die WNDCLASS-Struktur, die die Informationen über die Klasse empfängt. Bei Erfolg gibt *GetClassInfo* einen Wert ungleich 0 zurück. Wird keine passende Klasse gefunden, gibt *GetClassInfo* 0 zurück.

Siehe auch: Struktur *WNDCLASS*

GetInstance

```
HINSTANCE GetInstance () const;
```

Gibt das Instanz-Handle für dieses Modul zurück.

GetInstanceData

```
int GetInstanceData(void* data, int len) const;
```

GetInstanceData wird nur für 16-Bit-Anwendungen verwendet und holt Daten von einer bereits laufenden Instanz einer Anwendung. *len* ist die Größe des Puffers.

GetModuleFileName

```
int GetModuleFileName(char far* buff, int maxChars);
```

Gibt den erweiterten Dateinamen (Pfad und Dateiname) der Datei, aus der das angegebene Modul geladen wurde, zurück. *buff* zeigt auf einen Puffer, der Pfad und Dateinamen enthält. *maxChars* gibt die Länge des Puffers an. Der erweiterte Dateiname wird abgeschnitten, wenn er diese Länge überschreitet. *GetModuleFileName* gibt 0 zurück, wenn ein Fehler auftritt.

GetModuleUsage

```
int GetModuleUsage() const;
```

GetModuleUsage wird nur für 16-Bit-Anwendungen verwendet und gibt bei Erfolg den Referenzzähler des Moduls zurück. Der Referenzzähler wird bei jedem Aufruf von *GetModuleUsage* um eins erhöht und beim Aufruf von *FreeLibrary* um eins vermindert.

GetName

```
const char far* GetName() const;
```

Holt den Namen des Moduls.

Siehe auch: *::GetName* (Windows API)

GetParentObject

```
TWindow* GetParentObject(HWND hWndParent);
```

Holt ein Handle zum übergeordneten Fenster. Diese Funktion dient nur der Abwärtskompatibilität mit ObjectWindows 1.0.

Siehe auch: *::GetParentObject* (Windows API)

GetProcAddress

```
FARPROC GetProcAddress(const char far* fcnName) const;
```

Holt die Einsprungadresse der exportierten Prozedur oder Funktion *fcnName*, wenn es die Funktion findet; gibt andernfalls 0 zurück.

InitModule

```
void InitModule(HINSTANCE Instance, const char far* cmdLine);
```

Führt jede für das Modul erforderliche Initialisierung von Instanzen durch. Wenn das Modul nicht erzeugt werden kann, wird eine Exception *TXInvalidModule* ausgelöst.

IsLoaded

```
bool IsLoaded() const;
```

Gibt einen Wert ungleich 0 zurück, wenn das Instanz-Handle geladen ist. Um sicherzustellen, daß ein angegebenes Handle geladen ist, sollte in erster Linie diese Funktion verwendet werden.

LoadAccelerators

```
HACCEL LoadAccelerators(TResId id) const;
```

Lädt die durch *id* bezeichnete Tastenkürzeltabellen-Ressource. *LoadAccelerators* lädt die Tabelle nur dann, wenn sie nicht schon vorher geladen wurde, andernfalls vergibt *LoadAccelerators* ein Handle zur geladenen Tabelle.

LoadBitmap

```
HBITMAP LoadBitmap(TResId id) const;
```

Lädt die in *id* festgelegte Bitmap-Ressource. Wird die Bitmap nicht gefunden, gibt *LoadBitmap* 0 zurück.

Siehe auch: *TBitMap, TResId, ::LoadBitmap (Windows API), OBM_XXXX-Werte (Windows API)*

LoadCursor

```
HCURSOR LoadCursor(TResId id) const;
```

Lädt die durch *id* bezeichnete Cursor-Ressource in den Speicher und vergibt ein Handle. Wenn die Ressource nicht gefunden werden kann oder keine Cursor-Ressource ist, gibt *LoadCursor* 0 zurück.

Siehe auch: *TCursor, TResId, ::LoadCursor (Windows API)*

LoadIcon

```
HICON LoadIcon(const char far* name) const;
```

Lädt die durch den Parameter *name* bezeichnete Symbolressource in den Speicher. *LoadIcon* lädt das Symbol nur dann, wenn es nicht schon vorher geladen wurde, und gibt 0 zurück, wenn es die Ressource nicht findet.

Ein vordefiniertes Windows-Symbol kann mit *LoadIcon* geladen werden, wenn *name* auf einen der IDI_XXXX-Werte zeigt.

Siehe auch: *TIcon, ::LoadIcon (Windows API), IDI_XXXX-Werte (Windows API)*

LoadMenu

```
HMenu LoadMenu(TResId id) const;
```

Lädt die durch *id* bezeichnete Menüressource in den Speicher. Wenn es die Menüressource nicht findet, gibt *LoadMenu* 0 zurück.

Siehe auch: *TMenu, ::LoadMenu (Windows API)*

LoadResource

```
HGLOBAL LoadResource(HRSRC hRsrc) const;
```

Lädt eine Ressource - die durch *hRsrc* bezeichnet ist - in den Speicher und vergibt ein Handle zum Speicherblock, der die Ressource enthält. Gibt 0 zurück, wenn die Ressource nicht gefunden wird. Der *hRsrc*-Parameter muß ein durch *FindResource* erzeugtes Handle sein.

LoadResource lädt die Ressource nur dann in den Speicher, wenn sie nicht schon vorher geladen wurde. Ist die Ressource bereits geladen, erhöht

LoadResource den Referenzzähler um eins und vergibt ein Handle zu der vorhandenen Ressource. Die Ressource bleibt im Speicher, bis sie verworfen wird.

LoadString

```
int LoadString(uint id, char far* buff, int maxChars) const;
```

Lädt eine String-Ressource, die durch *id* festgelegt ist, in denjenigen Puffer, auf den *buff* zeigt. *maxChars* bestimmt die Größe des Puffers, in den der nullterminierte String kopiert wird. Ist der String länger als in *maxChars* festgelegt, wird er abgeschnitten. Der Rückgabewert ist die Anzahl der in den Puffer kopierten Zeichen oder 0, wenn die String-Ressource nicht existiert.

LoadString

```
string LoadString(uint id) const;
```

Lädt eine in *id* bezeichnete String-Ressource

LowMemory

```
bool LowMemory();
```

Die Funktion *LowMemory* ist veraltet. Sie gibt immer 0 zurück.

MakeWindow

```
TWindow* MakeWindow(TWindow* win);
```

Diese Funktion ist veraltet. Statt ihrer ist die Funktion *Create* aus *TWindow* zu verwenden.

operator HINSTANCE

```
operator HINSTANCE() const;
```

Liefert das Handle der Windows-Anwendung oder des DLL-Moduls zurück, die von diesem *TModule* vertreten werden. Das Handle muß Windows beim Laden von Ressourcen als Parameter übergeben werden.

operator ==

```
bool operator ==(const TModule& other) const;
```

Liefert **true** zurück, wenn diese Instanz der anderen gleicht; gibt andernfalls **false** zurück.

RestoreMemory

```
void RestoreMemory();
```

Diese Funktion, die veraltet ist, stellt Speicher wieder her.

SetInstance

```
void SetInstance(HINSTANCE hInstance);
```

Setzt das Instanz-Handle für dieses *TModule*. *SetInstance* wird in Sonderfällen verwendet, in denen *hInstance* beim Konstruieren des Moduls nicht bekannt ist.

SetName

```
void SetName(const char far* name);
```

Zugriffsfunktion, die den Namen des Moduls setzt.

SetResourceHandler

```
const RSRCHDLRPROC SetResourceHandler(const char far* type, RSRCHDLRPROC
                                      loadProc) const;
```

SetResourceHandler wird in 16-Bit-Anwendungen verwendet und installiert eine Callback-Funktion zum Laden von Ressourcen. *type* zeigt auf einen Ressourcen-Typ. *loadProc* ist die Adresse der Callback-Prozedur. Bei

erfolgreichem Aufruf gibt *SetResourceHandler* einen Zeiger auf einen bereits installierten Ressourcen-Handler zurück. Ist kein Ressourcen-Handler installiert, wird ein Zeiger auf den Standard-Ressourcen-Handler zurückgegeben. Diese Funktion ist hilfreich für die Verwaltung benutzerdefinierter Ressourcen-Typen.

SizeOfResource

```
uint32 SizeofResource(HRSRC hRsrc) const;
```

Gibt die Bytegröße der von *hRsrc* bezeichneten Ressource zurück. Die Ressource muß ein durch *FindResource* erzeugtes Ressourcen-Handle sein. Wird die Ressource nicht gefunden, ist der Rückgabewert 0.

Aufgrund der Ausrichtung in der ausführbaren Datei kann der Rückgabewert größer sein als die tatsächliche Größe der Ressource. Eine Anwendung sollte sich daher nicht auf diesen Wert als exakte Größe einer Ressource stützen.

ValidWindow

```
TWindow* ValidWindow(TWindow* win);
```

Diese Funktion, die veraltet ist, gibt ein Handle zum gültigen Fenster zurück.

Geschützte Datenelemente

HInstance

```
HINSTANCE HInstance;
```

Enthält die ausführende Instanz entweder der Windows-Anwendung oder des DLL-Moduls. Die Instanz muß Windows beim Laden von Ressourcen als Parameter übergeben werden.

Name

```
int GetModuleFileName(char far* buff, int maxChars);
```

Gibt den erweiterten Dateinamen (Pfad und Dateiname) der Datei an, aus der dieses Modul geladen wurde. *buff* zeigt auf einen Puffer, der den Pfad und den Dateinamen enthält. *maxChars* bezeichnet die Länge des Puffers. Der erweiterte Dateiname wird abgeschnitten, wenn er diese Länge überschreitet. *GetModeFileName* gibt 0 zurück, wenn ein Fehler auftritt.

Klasse TModule::TxInvalidModule — module.h

TXInvalidModule ist eine verschachtelte Klasse und beschreibt eine Exception, die durch ein ungültiges Modul entsteht. Ein Fenster löst diese Exception aus, wenn es kein gültiges *TModule*-Objekt erzeugen kann.

Öffentliche Konstruktoren

Konstruktor

```
TXInvalidModule();
```

Konstruiert ein *TXInvalidModule*-Objekt.

Öffentliche Elementfunktionen

Clone

```
TXOwl* Clone();
```

Legt eine Kopie des *TXInvalidModule*-Exception-Objektes an.

Throw

```
void Throw();
```

Löst ein *TXInvalidModule*-Exception-Objekt aus.

Klasse TOleClientDC olewindo.h

Wird aus *TClientDC* abgeleitet. *TOleClientDC* ist eine Hilfsklasse, die zwischen zwei verschiedenen Koordinatensystemen übersetzt. Die logischen Punkte des Fensters können z.B. in HIMETRIC oder *twips* gemessen werden, während die Koordinaten des eigentlichen Ausgabe-Gerätes (der Viewport) in Pixel gemessen werden. Ohne die Hilfe dieser Klasse müßte erst ein Client DC erzeugt werden, dann die logischen Koordinaten (der Ursprung) des Fensters, seine Höhe und Breite (die Ausmaße), sowie der Ursprung und das Ausmaß des Viewports (gemessen in Geräte-Koordinaten) gesetzt werden. Statt dessen erledigt *TOleClientDC* alle diese Berechnungen durch Abbildungen der logischen Punkte auf Gerätepunkte und umgekehrt.

TOleClientDC arbeitet mit einem Objekt *TOleWindow*. Standardmäßig führt *TOleClientDC* sowohl die Skalierung (Anpassen der Ausmaße des Fensters und des Viewports), als auch den Bildlauf (Anpassen des Ursprungs des Fensters und des Viewports) durch.

Öffentlicher Konstruktor

Konstruktor

```
TOleClientDC(TOleWindow& win, bool scale = true);
```

Konstruiert ein Objekt *TOleClientDC*. Der Parameter *win* verweist auf das von *TOleClientDC* zur Erzeugung eines DC benutzte Fenster. Wenn der Parameter *scale* **wahr** ist, dann erledigt *TOleClientDC* die Skalierung. Wenn Ihre Anwendung jedoch die Skalierung übernimmt, können sie *scale* als **false** übergeben.

Der Bildlauf wird durch die Anwesenheit eines Scrollers (*TScroller*) kontrolliert. Standardmäßig übernimmt *TOleClientDC* sowohl die Skalierung, als auch den Bildlauf.

Siehe auch: *TClientDC*, *TOleWindow*

Klasse TOleDocument oledoc.h

Wird aus *TStorageDocument* abgeleitet. *TOleDocument* implementiert die Dokumentenhälfte des DOk/View-Paares. Dies bedeutet, daß *TOleDocument* die Dokumentendaten bearbeitet, während das entsprechende Objekt *TOleView* bestimmt, wie die Daten auf dem Bildschirm dargestellt werden. Grundsätzlich ist *TOleDocument* ein *TStorageDocument* und verfügt durch seine Zeiger auf *TOcDocument* über Informationen über *TOcDocument*.

TOleDocument ist für die Erstellung von Verbunddokumenten (Dokumente, die eine Anzahl von verschiedenen eingebetteten Dokumenten von verschiedenen Quellanwendungen enthalten können), für das Schließen von Dokumenten, für das Einlesen von Dokumenten aus dem Lager (einem Bereich in einer Datei, in dem Daten gespeichert werden) und für das Schreiben der Dokumente in das Lager verantwortlich. Ist die Anwendung ein Server, dann besteht das Dokument aus einem einzigen Objekt. Im Falle eines Containers, kann das Dokument aus einem oder mehreren eingebetteten Objekten bestehen (diese werden auch als *Teile* bezeichnet).

Um diese Aufgaben zu bewältigen, kommuniziert *TOleDocument* durch die Nutzung von Funktionen wie *GetOcApp*, *GetOcDoc,* und *SetOcDoc* mit den zugrundeliegenden Klassen von ObjectComponents.

Siehe auch: *TOleView*, *TStorageDocument*, *TOcDocument*, *TOcRemView*

Öffentlicher Konstruktor und Destruktor

Konstruktor

```
TOleDocument(TDocument* parent = 0);
```

Erzeugt ein Objekt *TOleDocument*, das mit dem gegebenen übergeordneten Objekt *TDocument* assoziiert ist.

Destruktor

```
~TOleDocument();
```

Zerstört das Objekt *TOleDocument*. Im Falle eines OLE-Containers bleibt die Verbunddatei geöffnet, bis alle Ansichten geschlossen sind.

Öffentliche Elementfunktionen

CanClose

```
virtual bool CanClose();
```

Bereitet das Dokument auf das Schließen vor. Bevor das aktuelle Dokument geschlossen wird, wird überprüft, ob alle untergeordneten Dokumente geschlossen werden können. Wenn eines der untergeordneten Dokumente **false** zurückgibt, gibt auch *CanClose* **false** zurück und hält den Prozeß an. Wenn alle untergeordneten Dokumente **true** zurückgeben, prüft *CanClose*, ob das Dokument verändert wurde. Ist dies der Fall, wird der Benutzer gefragt, ob das Dokument abgespeichert, Änderungen verworfen oder die Operation abgebrochen werden soll. Wenn das Doku-

ment nicht verändert wurde und alle untergeordneten Dokumente **true** zurückgeben, gibt *CanClose* **true** zurück und zeigt damit an, daß das Dokument geschlossen werden kann.

CanClose ruft außerdem *ReleaseDo* für das assoziierte ObjectComponents-Dokument auf, um sicher zu stellen, das alle eingebetteten Objekte korrekt geschlossen wurden.

Siehe auch: *TOleDocument::ReleaseDoc*

Commit

```
bool Commit(bool force);
```

Überträgt die aktuellen Dokumentendaten in das Lager. Wenn *force* **true** ist und die Daten ein gültige Form haben, werden alle Daten in das Lager geschrieben und *Commit* gibt **true** zurück. Wenn *force* **false** ist, werden die Daten nur übertragen, wenn sie ungültig sind.

Die Daten wurden seit der letzten Speicheroperation verändert.

GetNewStorage

```
virtual IStorage*GetNewStorage();
```

Wird normalerweise in der Menüauswahl Speichern unter verwendet. *GetNewStorage* holt ein neues Lager für das Dokument.

Wenn der Pfad des Dokuments sich ändert, dann kann diese Funktion beispielsweise dazu verwendet werden, ein neues Lager zu erzeugen.

GetOcApp

```
TOcApp* GetOcApp();
```

Gibt die ObjectComponents-Anwendung, die mit diesem Objekt *TOleDocument* assoziiert ist, zurück.

Siehe auch: *TOcApp*

GetOcDoc

```
TOcDocument*GetOcDoc();
```

Gibt das ObjectComponents-Dokument, das mit diesem Objekt *TOleDocument* assoziiert ist, zurück.

Siehe auch: *TOleDocument::SetOcDoc*

InitDoc

```
InitDocvirtual bool InitDoc();
```

Überschreibt die Funktion *InitDoc* von *TDocument* und erzeugt oder öffnet eine Verbunddatei, damit es mit den eingebetteten Objekten dieses Dokuments ein assoziiertes *IStorage* gibt. Verwendet ein Objekt *TOcDocument*, um die eigentlichen Interaktionen mit den OLE-Schnittstellen *IStorage* und *IStream* , die schließlich für die Herstellung einer Beziehung zwischen einer Verbunddatei und ihrem Lager verantwortlich sind, auszuführen.

Siehe auch: *TOcDocument*, *TDocument::InitDoc*

Open

```
bool Open(int mode, const char far* path);
```

Lädt die eingebetteten Objekte, wenn welche vorhanden sind, mittels des angegebenen Pfades *path*. *mode* ist eine Kombination von Bits, die angeben, wie die eingebetteten Objekte zu öffnen sind (z.B. Nur lesen, Lesen/Schreiben, usw.).

PathChanged

```
bool PathChanged();
```

Überprüft, ob der Pfad des aktuellen Dokuments derselbe Pfad wie der von *TOcDocument* ist. Wenn die Pfade nicht übereinstimmen, übergint *PathChanged* **true**.

Siehe auch: *TOcDocument*

PreOpen

```
virtual void PreOpen();
```

Bevor das Dokument geöffnet wird, gibt *PreOpen* der abgeleiteten Klasse die Möglichkeit, eine bestimmte Operation, wie z.B. das Setzen eines anderen Öffnugsmodus für das Verbunddokument, auszuführen.

Siehe auch: dtxxxx document constants

Read

```
virtual bool Read();
```

Lädt die eingebetteten Objekte aus der Verbunddatei. Ein Container sollte diese Funktion aufrufen, wenn er eingebettete Objekte aufrufen möchte.

Siehe auch: TOleDocument::Write

ReleaseDoc

```
Virtual bool RelaseDoc()
```

Gibt das ObjectComponents-Dokument frei, sobald der Server mit der Verwendung des Dokuments fertig ist.

Siehe auch: *TOleDocument::CanClose*

SetOcDoc

```
void SetOcDoc(TOcDocument* doc);
```

Stellt das ObjectComponents-Dokument ein, das mit diesem Objekt *TOleDocument* assoziiert ist.

Siehe auch: *TOleDocument::GetOcDoc*

SetStorage

```
virtual bool SetStorage(IStorage* stg);
```

Gliedert den Zeiger (*stg*) *IStorage* diesem Dokument an. Bei Erfolg gibt *SetStorage* **true** zurück.

Eine OLE Lager-Schnittstelle, die ObjectComponents implementiert, wenn einem Dokument ein Lager zugeteilt werden soll.

Write

```
virtual bool Write();
```

Speichert die eingebetteten Objekte in der Verbunddatei ab. Ein Container sollte diese Funktion aufrufen, sobald seine eingebettetens Objekte in einem Lager abgespeichert werden sollen.

Siehe auch: *TOleDocument::Read*

Klasse TOleFactoryBase<> olefacto.h

Eine Template-Klasse. *TOleFactoryBase*<> erzeugt einen Rückrufcode für ObjectWindows-Klassen. Der Hauptzweck des Factory-Codes ist es, eine Rückruf-Funktion *Create* zu liefern, die ObjectComponents aufruft, um Objekte zu erzeugen.

So wie eine Kochrezept aus einer Liste von Anweisungen zum Kochen eines bestimmten Essens besteht, enthält ein Template, wie z.B. *TOleFactoryBase*<> Anweisungen, wie ein Objekt erstellt werden soll, in diesem Fall ein Factory-Objekt. *TOleFactoryBase*<> enthält zwei öffentliche Elementfunktionen. Die drei weiteren Funktionen werden als Template-Argumente weitergegeben. Obwohl diese Template-Argumente eigentlich zu der Klasse gehören, die sofort aufgerufen wird, wenn die Argumente in *TOleFactoryBase*<> eingesetzt werden, werden Sie hier der Einfachheit halber beschrieben.

Benutzen Sie *TOleFactoryBase*<>, um Objekte im Allgemeinen herzustellen, unabhängig davon, ob sie eingebettet sind oder nicht, OLE-Fähigkeiten besitzen oder das Doc/View Modell verwenden. Diese Objekte können, müssen aber nicht mit OLE verbunden sein.

Die Aufrufe werden durch die an Template-Klassen weitergegebenen Argumente zur Verfügung gestellt. Die Factory-Base-Klasse verwendet drei Template-Parameter: den Anwendungstyp, eine Menge von Funktionen, um das Objekt zu erzeugen und eine Menge von Funktionen, um ein automatisiertes Objekt zu erzeugen. In Abhängigkeit von den weitergegebenen Argumenten können folgende OLE-aktivierten Komponenten erstellt werden:

- automatisierte Doc/View-Komponenten
- nicht automatisierte Doc/View-Komponenten
- automatisierte Nicht-Doc/View-Komponenten
- nicht automatisierte Nicht-Doc/View-Komponenten

ObjectWindows liefert eine Standard-Implementation für die Objekterzeugung und die Objektautomatisierung. Factory-Template-Klassen geben einen Überblick über diese Klassen.

Durch die Verwendung von *TOleFactoryBase*<>, um eine OLE-Schnittstelle für Ihre Anwendung zu erhalten, auf die OLE zugreifen kann. Dies bedeutet, das *TOleFactoryBase*<> jede Beziehung mit *IUnknown* behandelt, einer Standard-Schnittstelle von OLE.

Siehe auch: Typdefinition *TComponentFactory*, Klasse *TOcRegistrar*, Klasse *TAutoFactory*

Öffentliche Elementfunktionen> Klasse;öffentliche Elementfunktionen>

TComponentFactory>

```
operator TComponentFactory();
```

Konvertiert das Objekt in einen Zeiger auf der Factory. ObjectComponents benutzt diesen Zeiger, um das Objekt zu erzeugen.

Create>

```
static IUnknown* Create(IUnknown* outer, uint32 options, uint32 id);
```

Eine Rückruffunktion *TComponentFactory*, die die Anwendung erzeugt oder zerstört oder Objekte erzeugt. Wenn ein Anwendungsobjekt nicht bereits exisitiert, dann erzeugt *Create* ein neues Anwendungsobjekt. Das Argument *outer* zeigt auf die OLE2-Schnittstelle *IUnknown*, mit dem sich dieses Objekt selbst zusammenfaßt. Wenn *outer* 0 ist, wird das Objekt zu einem unabhängigen Objekt.

Das Argument *options* zeigt den Modus der Anwendung zur Laufzeit an. Die Werte für *options* werden entweder von der Befehlszeile oder von ObjectComponents gesetzt. Sie werden durch *Registrar* an diesen Rückruf weitergegeben. Die Anwendung erfährt durch diese Flaggen, wie sie sich zu verhalten hat, und der Factory-Rückruf erfährt, was zu tun ist. Beispielsweise zeigt der Wert von *amExeMode* an, daß der Server als ein .EXE läuft, weil er entweder als .EXE erzeugt wurde, oder weil er eine .DLL ist, die durch eine .EXE gestartet wurde und jetzt als ausführbares Programm läuft. Die Darstellung *TOcAppMode enum* zeigt die möglichen Werte für das Argument *options* an.

Wenn die Anwendung bereits existiert und die Objekt-ID (*id*) gleich 0 ist, dann gibt *Create* die OLE-Schnittstelle der Anwendung zurück. Andernfalls ruft *Create OCInit* auf, um ein neues *TOcApp* zu erzeugen und um die Optionen von *TOcAppMode enum* zu registrieren, das OLE verwandte Flaggen zur Verwendung in der Befehlszeile der Anwendung enthält. (Diese Flaggen sagen der Anwendung, ob sie als Server gelaufen ist, ob sie sich selbst registrieren muß, usw.) Wenn eine Komponenten-ID weitergegeben wurde, dann wird diese zur Komponente; ansonsten startet *Create* die Anwendung selbst, basierend auf den Werten von *TOcAppMode enum*, und gibt die OLE-Schnittstelle zurück.

Siehe auch: *TOleFactoryBase::DestroyApp*, *TOcAppMode enum*

Template-Argumente> Klasse;Template>

CreateApp

```
static T* CreateApp(options);
```

Erzeugt eine neue Anwendung. Standardmäßig erzeugt es eine neue Anwendung des Template-Typs *T* ohne Argumente. Dies ist eine statische Funktion, die Sie in Ihrer Anwendung überschreiben können.

Die *options* sind die, die zu der Factory übergeben werden. Sie können einen der Werte von *TOcAppMode enum* annehmen (z.B. *amRun*, *amEmbedding*, u.s.w.), die den Modus der Anwendung zur Laufzeit angeben und die Optionen an die Factory weiterleiten.

Siehe auch: *TOleFactoryBase::DestroyApp*

CreateObject

```
static IUnknown* CreateObject(TApplication* app, TDocTemplate* tpl,
IUnknown* outer);
```

Erzeugt ein Objekt unter Verwendung der Dokument-Templates, auf die in *tpl* Bezug genommen wird, und der in *app* angegebenen Anwendung. Der Parameter *outer* bezieht sich auf die kontrollierende Schnittstelle *IUnknown* des Objekts, mit dem dieses Objekt zusammengefaßt werden soll. Wenn *outer* 0 ist, dann ist das Objekt unabhängig.

Diese statische Funktion kann überschrieben werden, damit Sie Ihr eigenes Objekt zur Laufzeit erzeugen können.

Siehe auch: *TOleFactoryBase::DestroyApp*, Klasse *TOcView*

DestroyApp

```
static void DestroyApp(T* app);
```

Zerstört die Anwendung (*app*), indem die Registrierung des Objektes entfernt und das Objekt gelöscht wird.

Siehe auch: *TOleFactoryBase::CreateApp*

Klasse TOleFrame — oleframe.h

Wird aus *TDecoratedFrame* abgeleitet. *TOleFrame* liefert Unterstützung für die Benutzeroberfläche des Hauptfensters einer Single Document Interface-Anwendung (SDI). Da es die Funktionalität von *TDecoratedFrame* geerbt hat, kann *TOleFrame* Dekorationen wie beispielsweise Mauspaletten um das Client-Fenster plazieren. Wegen seiner OLE-Rahmenfunktionalität sollte *TOleFrame* als ein Hauptrahmen erzeugt werden. *TOleFrame* unterstützt beispielsweise grundlegende Container-Operationen wie

- Platz in einem Container-Rahmenfenster erzeugen, den der Server verlangt hat
- Das Servermenü mit dem Containermenü zusammenführen
- Beschleuniger und andere Botschaften aus der Warteschlange des Servers verarbeiten

Zusätzlich zur Unterstützung der üblichen Operationen und Ereignisbehandlungen für Rahmenfenster stellt *TOleFrame* Funktionen zur Verfügung, die das Zusammenstellen für Popup-Menüs von OLE 2-Menüs unterstützen.

Durch die Verwendung der Elementfunktion EvOcXxxx, die Ereignisse behandelt, antwortet *TOleFrame* auf ObjectComponents-Botschaften, die sowohl an Container-, als auch an Server-Anwendungen geschickt werden. Obwohl die meisten Botschaften und Funktionen Container-Unterstützung liefern, bezieht sich eine Funktion, und zwar *EvOcAppShutDown*, auf den Server, und eine Funktion, nämlich *GetRemViewBucket*, ermöglicht die Unterstützung des Servers. Ob *TOleFrame* als Server oder als Container funktioniert, es besitzt immer einen Zeiger zu einem korrespondierenden *TOcApp*.

Siehe auch: *TOcApp*, *TDecoratedFrame*

Öffentlicher Konstruktor und Destruktor

Konstruktor

```
TOleFrame(const char far* title, TWindow* clientWnd, bool
trackMenuSelection = false, TModule*  module = 0);
```

Konstruiert ein Objekt *TOleFrame* mit den angegebenen Titeln für das Rahmenfenster (*title*), das Client-Fenster (*ClientWnd*) und der Modulinstanz. Der Parameter *trackMenuSelection* zeigt an, ob das Rahmenfenster Menüauswahlpunkte aufzeichnen soll (d.h. Hilfetext in der Stausleiste darstellen soll, sobald ein Menüeintrag hervorgehoben wird) oder nicht.

Destruktor

```
~TOleFrame();
```

Zerstört ein Objekt *TOleFrame*.

Öffentliche Elementfunktionen

AddUserFormatName

```
void AddUserFormatName(char far* name, char far* resultName, char far*
id);
```

Fügt durch den Benutzer definierte Formate und die korrespondierenden Namen der Liste der Formate der Zwischenablage hinzu. Benutzen Sie diese Funktion, um ein Datenformat der Zwischenablage (*name*) mit der Beschreibung des Datenformats, wie es den Benutzern im Hilfetext des Dialogfensters Manuell einfügen (resultName) erscheint, zu verbinden. Um ein Standardformat der Zwischenablage zu verwenden, setzen Sie den Parameter *id* auf eine entsprechende Konstante (z.B. CF_TEXT). Ansonsten übergeben Sie die Zeichenkette als Name und lassen die ID weg, falls das Format durch eine Zeichenkette identifiziert wird.

Siehe auch: *TOcApp::AddUserFormatName*

GetOcApp

```
TOcApp* GetOcApp();
```

Holt die ObjectComponents-Anwendung, die mit diesem Rahmenfenster in Verbindung steht.

Siehe auch: *TOcApp*, *TOleFrame::SetOcApp*

GetRemViewBucket

```
TWindow* GetRemViewBucket();
```

Gibt einen Zeiger zum versteckten Helper-Fenster des OLE-Rahmens zurück, das alle inaktiven Server-Fenster festhält. Es kann außerdem direkte Mauspaletten und Fenster vom Typ *TOleView* enthalten.

SetOcApp

```
void SetOcApp(TOcApp* app);
```

Ändert die mit diesem Rahmenfenster verknüpfte ObjectComponents-Anwendung in die Anwendung, die im Parameter *app* angegeben ist.

Siehe auch: *TOcApp*, *TOleFrame::GetOcApp*

Geschützte Elementfunktionen

CanClose

```
bool CanClose();
```

Übergibt **true**, wenn das Rahmenfenster geschlossen werden kann. Überprüft, ob sowohl *TOcApp*, als auch alle untergeordneten Fenster geschlossen werden können. Wenn die Anwendung und alle untergeordneten Fenster **true** zurückgeben, schließt *CanClose* das Rahmenfenster.

Siehe auch: *TOcApp*

CleanupWindow

```
void CleanupWindow();
```

Führt eine normale Fensterbereinigung jeder HWND-bezogenen Ressource durch. Für DLL-Server zerstört *CleanupWindow* den aktiven Zeitgeber.

Siehe auch: *TWindow::CleanupWindow*, *TOleFrame::EvTimer*

Destroy

```
void Destroy(int retVal);
```

Stellt mit allen verbundenen Servern sicher, daß sie geschlossen werden können, bevor das Rahmenfenster zerstört wird. Wenn der Benutzer die Anwendung mit noch immer eingebetteten Objekten schließt, dann versteckt *Destroy* das Rahmenfenster, anstatt es zu zerstören.

DestroyStashedPopups

```
void DestroyStashedPopups();
```

Zerstört alle vorher gespeicherten und gemeinsam genutzten Popup-Menüs. Überprüft, ob *StashCount* 0 ist, bevor die Menüs zerstört werden.

Siehe auch: *TOleFrame::StashContainerPopups*, *TOleFrame::StashCount*

EvActivateApp

```
void EvActivateApp(bool active, HTASK hTask);
```

Antwortet auf die Botschaft WM_ACTIVATEAPP, die gesendet wird, sobald ein Fenster aktiviert oder deaktiviert wird. Wenn *active* **true** ist, wird das Fenster aktiviert.

Diese Botschaft wird an das höchste Fenster, das deaktiviert wird, geschickt, bevor sie an das höchste Fenster, das deaktiviert wird, geht. *hTask* ist eine Kennung des aktuellen Prozesses.

Dieses Ereignis wird an das Objekt *TOcApp*, das einen In-Place Server aktiviert, sofern dieser existiert, weitergegeben.

Siehe auch: *TOcApp*

EvOcAppBorderSpaceReq `bool EvOcAppBorderSpaceReq(TRect far* rect);`

Antwortet auf die Nachricht OC_APPBORDERSPACEREQ, die an einen Container geschickt wurde. Die Antwort ist, den Container zu fragen, ob er abgegrenzten Raum in seinem Rahmenfenster dem Server zur Verfügung stellen kann.

Siehe auch: *TOcApp::BorderSpaceSet*

EvOcAppBorderSpaceSet `bool EvOcAppBorderSpaceSet(TRect far* rect);`

Antwortet auf die Nachricht OC_APPBORDERSPACESET, indem es im Rahmenfenster des Containers Platz für den abgegrenzten Platz, den der Server angefordert hat, schafft.

Siehe auch: *TOcApp::BorderSpaceReq*

EvOcAppDialogHelp `void EvOcAppDialogHelp(TOcDialogHelp far& dh);`

Antwortet auf die Nachricht OC_APPDIALOGHELP. Der Parameter *dh* bezieht sich auf eine der Konstanten *TOcDialogHelp enum*, die anzeigen, welche Art von Dialogfenster der Benutzer geöffnet hat. Beispielsweise zeigt *dhBrowseLinks* an, daß das linke Dialogfenster geöffnet ist. Das *TOcDialogHelp enum* zählt die Hilfekonstanten und deren äquivalenten Werte ihrer Dialogfenster auf.

Siehe auch: *TOcDialogHelp enum*

EvOcAppFrameRect `bool EvOcAppFrameRect(TRect far* rect);`

Antwortet auf die Nachricht OC_APPFRAMERECT, die an einen Container geschickt wurde. Die Antwort ist, die Koordinaten des Client-Bereichsrechtecks des Hauptfensters der Anwendung zu holen.

EvOcAppInsMenus `bool EvOcAppInsMenus(TOcMenuDescr far& sharedMenu);`

Antwortet auf die Nachricht OC_APPINSMENUS, indem es das Menü des Containers in das gemeinsam genutzte Menü einsetzt. Der Parameter *sharedMenu* bezieht sich auf dieses zusammengesetzte Menü.

EvOcAppMenus `bool EvOcAppMenus(TOcMenuDescr far& md);`

Antwortet auf die an den Container geschickte Nachricht OC_OCAPPMENUS. Als Antwort wird eine zusammengesetzte Menüleiste installiert.

EvOcAppProcessMsg `bool EvOcAppProcessMsg(MSG far* msg);`

Antwortet auf die Nachricht OC_APPROCESSMSG, die an einen Container mit der Anfrage geschickt wurde, Beschleuniger und andere Nachrichten aus der Nachrichtenschlange des Containers zu verarbeiten.

EvOcAppRestoreUI

```
void EvOcAppRestoreUI();
```

Antwortet auf die Nachricht OC_APPRESTOREUI, indem sie das normale Containermenü und die normalen Containergrenzen wiederherstellt, sobald die direkte Bearbeitung beendet wird.

EvOcAppShutdown

```
bool EvOcAppShutdown();
```

Antwortet auf die Nachricht OC_APPSHUTDOWN, die anzeigt, daß das zuletzt eingebettete Objekt geschlossen wurde. Als Antwort wird der Server geschlossen.

EvOcAppStatusText

```
void EvOcAppStatusText(const char far*);
```

Antwortet auf die Nachrich OC_APPSTATUSTEXT, indem sie Text aus dem Server in der Statusleiste dieses Containers darstellt.

EvOcEvent

```
LRESULT EvOcEvent(WPARAM wParam, LPARAM lParam);
```

Antwortet auf die WM_OCEVENT Nachricht und leitet sie an eine der ereignisbehandelnden Funktionen EvOcXxxx, die auf dem Parameter *wParam* basieren, weiter. Nachrichten des Typs WM_OCEVENT werden von ObjectComponents abgeschickt, wenn es mit einem von OLE generierten Ereignis kommunizieren muß, z.B. wenn ein Server Mauspaletten darstellen soll.

EvSize

```
void EvSize(uint sizeType, TSize& size);
```

Antwortet auf die Nachricht EV_WM_SIZE, die eine Änderung der Größe des Rahmenfensters anzeigt und diese Information an *TOcApp* weitergibt, die dann jeden In-Place Server benachrichtigt. Der Server benutzt diese Information, um die Größe seiner Mauspalette zu ändern, sofern dies nötig ist.

Siehe auch: *TOcApp::EvActivate*

EvTimer

```
void EvTimer(uint timerId);
```

Handelt es sich um einen DLL-Server, dann antwortet *EvTimer* auf eine Timer-Botschaft, indem sie eine leere Schleife durchläuft, wenn die Nachrichtenschlange leer ist.

Siehe auch: *TOleFrame::CleanupWindow*

SetupWindow

```
void SetupWindow();
```

Verknüpft die ObjectComponents-Anwendung mit dem HWND des Fensters, damit *TOcApp* und das Fenster miteinander kommunizieren können.

Bereitet einen Platz für die Mauspalette des Servers vor, die eingefügt wird, sobald die direkte Bearbeitung des eingebetteten Objektes verlangt wird.

Siehe auch: *TOcApp*

StashContainerPopups `void StashContainerPopups(const TMenuDescr& shMenuDescr);`

Speichert eine lokale Kopie der Popup-Menüs ab, damit diese zur Menüzusammensetzung genutzt und später durch *DestroyStashedPopups* zerstört werden können. *shMenuDescr* ist die gemeinsam genutzte Menüdarstellung, die abgespeichert werden muß. Sie erhöht *StashCount* jedesmal, wenn Pop-Up Menüs abgespeichert werden.

Siehe auch: *TOleFrame::DestroyStashedPopups*, *TMenuDescr*

Geschützte Datenelemente

HoldMenu `HMENU HoldMenu;`

Enthält die Kennung der vorher gespeicherten Kopie des Containermenüs.

OcApp `TOcApp* OcApp;`

Zeigt auf die mit diesem Rahmenfenster verknüpfte ObjectComponents-Anwendung.

Siehe auch: *TOcApp*

StashCount `int StashCount;`

Enthält die Zahl der Menüleisten, die gespeichert worden sind. Diese Zahl gibt an, wieviele aktive direkte Bearbeitungssitzungen Sie zur Zeit geöffnet haben.

Siehe auch: *StashContainerPopups*, *TOleFrame::DestroyStashedPopups*

StashedContainerPopups `TMenu StashedContainerPopups;`

Enthält die gespeicherten und gemeinsam genutzten Popup-Menüs.

Siehe auch: *StashContainerPopups*, *TOleFrame::StashCount*

Einträge in der Anworttabelle

Einträge in die Antworttabelle	Elementfunktion
EV_WM_SIZE	EvSize
EV_WM_ACTIVATEAPP	EvActivateApp
EV_MESSAGE(WM_OCEVENT, EvOcEvent)	EvOcEvent
EV_OC_APPINSMENUS	EvOcAppInsMenus
EV_OC_APPMENUS	EvOcAppMenus

EV_OC_APPROCESSMSG	EvOcAppProcessMsg
EV_OC_APPFRAMERECT	EvOcAppFrameRect
EV_OC_APPBORDERSPACEREQ	EvOcAppBorderspaceReq
EV_OC_APPBORDERSPACESET	EvOcAppBorderSpaceSet
EV_OC_APPSTATUSTEXT	EvOcAppStatusText
EV_OC_APPRESTOREUI	EvOcAppRestoreUI
EV_OC_APPSHUTDOWN	EvOcAppShutDown

Klasse TOleMDIFrame olemdifr.h

Abgeleitet von *TMDIFrame* und *TOleFrame*. *TOleMDIFrame* liefert Unterstützung für das HAuptfenster der OLE-Benutzeroberfläche einer Multiple Document Interface-Anwendung (MDI). *TOleMDIFrame* kommuniziert durch die Nutzung eines Zeigers auf das Objekt *OcApp* direkt mit den ObjectComponents-Klassen.

TOleMDIFrame erbt die Funktionalität von *TMDIFrame*, die die Verwendung von MDI-Rahmenfenstern unterstützt. Diese Rahmenfenster wurden so entworfen, daß sie als Hauptfenster einer MDI-fähigen Anwendung dienen. Von *TOleFrame* erbt *TOleMDIFrame* dekorierende Rahmenfenster-Funktionalität, die das Hinzufügen von Dekorationen (wie Mauspaletten und Statusleisten) zu Rahmenfenstern unterstützt. *TOleMDIFrame* erbst auch die Fähigkeit,

- Platz zu erzeugen, den der Server in einem Container-Rahmenfenster angefordert hat.
- Container-Menüs in die Server-Menüs einzusetzen.
- Beschleuniger und andere Botschaften aus der Server-Warteschlange zu verarbeiten.
- OLE 2-Menüzusammensetzung für Popup-Menüs zu unterstützen.

TOleMDIFrame erbt von *TOleFrame* außerdem die Fähigkeit, durch Nutzung eines Zeigers auf das Objekt *OcApp* direkt mit den ObjectComponents-Klassen zu kommunizieren.

Der Schritt 14 im *ObjectWindows Handbuch* zeigt ein Beispielprogramm, das *TOleMDIFrame* zur Erzeugung eines OLE-fähigen dekorierten MDI-Rahmenfensters verwendet.

Siehe auch: *TMDIFrame*, *TDecoratedMDIFrame*, *TOleFrame*, *TOleFrame::SetOcApp*

Öffentlicher Konstruktor und Destruktor

Konstruktor

```
TOleMDIFrame(const char far* title, TResIdmenuResId, TMDIClient&
clientWnd = *new TMDIClient,
   bool trackMenuSelection = false, TModule*module = 0);
```

Konstruiert ein Objekt *TOleMDIFrame* mit dem angegebenen Titel, Ressourcen-ID des Menüs, Client-Fenster und der Modulinstanz.

Standardmäßig wird der Hinweistext des Menüs nicht abgezeigt, weil *trackMenuSelection* **false** ist (Diese Parameter stimmen mit denen des Konstruktors *TMDIFrame* überein.)

Destruktor

```
~TOleMDIFrame();
```

Zerstört das OLE-MDI-Rahmenfensterobjekt.

Siehe auch: *TDecoratedMDIFrame*

Geschützte Elementfunktionen

DefWindowProc

```
LRESULT DefWindowProc(uint message, WPARAM wParam, LPARAM lParam);
```

Erlaubt standardmäßiges Verarbeiten aller Nachrichten außer für eine die Größe verändernde Botschaft, die das Rahmenfenster betrifft. In diesem Fall gibt *DefWindowProc* nichts zurück.

EvActivateApp

```
void EvActivateApp(bool active, HTASK hTask);
```

Antwortet auf eine Nachricht, die anzeigt, daß das Rahmenfenster dieser Anwendung (*hTask*) entweder aktiviert (*active* ist **true**) oder deaktiviert (*active* ist **false**) werden wird. Außerdem gibt es diese Informationen an das Objekt *TOcApp* weiter.

Siehe auch: *TOcApp*

EvOcAppInsMenus

```
bool EvOcAppInsMenus(TOcMenuDescr far*);
```

Fügt neue Menüs in eine zur Verügung gestellte Menüleiste ein oder setzt diese mit denen eines untergeordneten Fensters oder Servers zusammen. Dazu erzeugt *EvOcAppInsMenus* zusammengesetzte temporäre Menüs für das untergeordnete Rahmen- und MDI-Fenster und kopiert dann die Breite des gemeinsam genutzten Menüs in die ObjectComponents-Struktur. Sie speichert die Container-Popup-Menüs, damit diese später gelöscht werden können.

Einträge in der Anworttabelle

Eintrag der Antworttabelle	Elementfunktion
EV_WM_ACTIVATEAPP	*EvActivateApp*
EV_OC_APPINSMENUS	*EvOcAppInsMenus*

Klasse TOleView oleview.h

Aus *TWindowView* und *TView* abgeleitet. *TOleView* unterstützt die View-Hälfte eine Dok/View-Paares und erzeugt ein Fenster mit einer Ansicht, die ein assoziiertes Dokument darstellen kann. Dokumente verwenden Ansichten, um sich selbst einem Benutzer darzustellen. Egal, ob eine Ansicht zu einem Server oder einem Container gehört, *TOleView* stellt das korrespondierende Objekt *TOcDocument* (ein ganzes zusammengefaßtes Dokument) ein.

Im Fall einer OLE-fähigen Container-Anwendung bezieht sich *view* auf das Fenster, indem die Container-Anwendung das Verbunddokument zeichnet, das aus einem oder mehreren verknüpften oder eingebetteten Objekten besteht . Um diese Objekte in verschiedenen Formaten darzustellen, kann ein Container mit mehr als einer Ansicht verknüpft sein. Ähnlich verhält es sich, wenn Daten richtig dargestellt werden sollen. Jedes eingebettete Objekt kann seine eigene Ansicht besitzen. Jede Containeransicht erzeugt ein korrespondierendes ObjectComponents- Objekt *TOcView*.

Wenn die Ansicht zu einer OLE-befähigen Server-Anwendung gehört, erzeugt *TOleView* eine Fernansicht des Server-Dokuments (ein Objekt *TOcRemView*). *TOleView* überträgt Nachrichten vom Server an den Container, insbesondere im Fall von zusammengestzten Menüs und Wiederherstellen von eingebetteten Objekten. *TOleView* unterstützt die Zusammensetzung des Server- und Containermenüeinträge, um ein zusammengesetztes Menü zu erzeugen. Da *TOleView* die Dimensionen des Server-Views kennt, ist *TOleView* dafür verantwortlich, dem Container mitzuteilen, wie das eingebettete Objekt wiederherzustellen ist.

Ähnlich wie *TView*, unterstützt *TOleView* die Erstellung von Ansichten und stellt mehrere Funktionen zur Ereignisbehandlung zur Verfügung, die dem View das Abfragen, Übergeben und Schließen von Ansichten ermöglichen. *TOleView* übernimmt es außerdem das Schreiben von Dokumenten in das Lager, die zu einem Container oder Servergehören.

Siehe auch: *TOcDocument*, *TOcView*, *TOcRemView*, *TOleWindow*, *TView*

Öffentliche Konstruktoren und Destruktoren

Konstruktor

```
TOleView(TDocument& doc, TWindow* parent = 0);
```

Konstruiert ein Objekt vom Typ *TOleView*, das mit dem gegebenen Dokumentenobjekt (*doc*) und dem übergeordneten Fenster (*parent*) verknüpft ist.

Destruktor

```
~TOleView();
```

Zerstört das Objekt *TOleView* und löst die Ansicht vom verknüpften Dokument ab.

Öffentliche Elementfunktionen

GetViewName

```
const char far* GetViewName();
```

Überschreibt die virtuelle Funktion *GetViewName* von *TView* und übergibt den Namen der Klasse (*TOleView*).

Siehe auch: *TView::GetViewName*

GetWindow

```
TWindow* GetWindow();
```

Überschreibt die virtuelle Funktion *GetWindow* von *TView* und übergibt die Instanz *TWindow*, die mit diesem View assoziiert ist.

Siehe auch: *TView::GetWindow*

SetDocTitle

```
bool SetDocTitle(const char far* docname, int index);
```

Überschreibt die virtuelle Funktion *TWindow* von *TView* und speichert den Titel des Dokuments, das mit dieser Ansicht assoziiert ist.

Siehe auch: TView::SetdocTitle

StaticName

```
static const char far* StaticName();
```

Übergibt die konstante Zeichenkette *"Ole View"*, die im Auswahlfenster der Benutzeroberfläche dargestellt wird.

Siehe auch: *TListView::StaticName*

Geschützte Elementfunktionen

CanClose

```
bool CanClose();
```

Eine Ansicht verwendet diese Funktion, um zu kontrollieren, ob sie geschlossen werden kann oder nicht. Wenn es sich um eine Serveransicht handelt, überprüft *CanClose*, ob in einem der eingebetteten Objekte eine offene Editiersitzung vorkommt. Wenn ja, schließt *CanClose* diese Sitzung, indem es das eingebettete Objekt von seinem Server trennt. Sie versteckt das Rahmenfenster des Servers und gibt gegebenenfalls **true**

zurück. Wenn es sich um einen Container handelt, fragt *CanClose* alle Dokumente und Ansichten ab und übergibt **true**, wenn alle geschlossen werden können.

Anders als bei der direkten Bearbeitung, die im Container-Fenster stattfindet, tritt offenes Editieren im Serverfenster auf.

CreateOCView

```
TOcView* CreateOCView(TDocTemplate* tpl, bool isEmbedded, IUnknown* outer);
```

Erzeugt eine mit dem eingebetteten Objekt assoziierte Ansicht. Verknüpft die Ansicht mit der in *tpl* angegebenen Dokument-Template. Der Parameter *isEmbedded* ist wahr, wenn die Ansicht ein eingebettetes Objekt ist. Der Parameter *outer* bezieht sich auf die Schnittstelle *IUnknown*, mit dem sich die Ansicht zusammenschließen will.

EvOcViewAttachWindow

```
bool EvOcViewAttachWindow(bool attach);
```

Verknüpft diese Ansicht mit ihrem übergeordneten Fenster, damit das eingebettete Objekt entweder geöffnet und editiert oder deaktiviert werden kann. Um eine Ansicht mit einem eingebetteten Objekt zu verknüpfen, wird der Parameter *attach* auf **true** gesetzt. Um die Verknüpfung aufzuheben, wird der Parameter auf **false** gesetzt.

EvOcViewClose

```
bool EvOcViewClose();
```

Befiehlt dem Server, die mit diesem Dokument verknüpfte Ansicht zu schließen. Überprüft, ob das Dokument seit dem letzten Speichervorgang verändert wurde. Übergibt **true**, wenn das Dokument und seine assoziierte Ansicht geschlossen sind.

Siehe auch: *TOleView::EvOcViewSavePart*

EvOcViewInsMenus

```
bool EvOcViewInsMenus(TOcMenuDescr far& sharedMenu);
```

Fügt das Severmenü in das zusammengefasste Menü ein. Bestimmt die Anzahl der Gruppen und Popup-Menüeinträge, die innerhalb einer jeden Gruppe einzufügen sind. Das gemeinsam genutzte Menü (*sharedMenu*) ist das mit den Server-Menügruppen zusammengefaßte Containermenü.

Siehe auch: *TMenuDescr* enthält weitere Informationen über das Zusammenfassen von Menüs, Struktur *TOcMenuDescr*

EvOcViewLoadPart

```
bool EvOcViewLoadPart(TOcSaveLoad far& ocLoad);
```

Befiehlt dem Server, sich selbst aus dem Lager zu laden. Lädt das Dokument und die verknüpfte Ansicht.

EvOcViewOpenDoc

```
bool EvOcViewOpenDoc(const char far* path);
```

Befiehlt der Containeranwendung, ein existierendes Dokument zu öffnen, damit das Dokument verbundene und eingebettete Objekte in Empfang nehmen kann. (*TOleView* wendet sich eigentlich an das Objekt

TOleDocument, um das Dokument aus dem Lager zu lesen, indem die OLE Standard-Schnittstellen *IStorage* und *IStream* genutzt werden.)

Gibt dem Dokument eine eindeutug bestimmte Kennungszeichenkette und übergibt **true**, wenn die Funktion erfolgreich war.

Siehe auch: *TOleView::EvOcViewClose*

EvOcViewPartInvalid

```
bool EvOcViewPartInvalid(TOcPart far& part);
```

Benachrichtigt die aktive Ansicht von allen Änderungen, die an den Daten des eingebetteten Objekts (*parts*) vorgenommen wurden. Benachrichtigt alle anderen mit diesem Dokument assoziierten Ansichten, daß das abgrenzende Rechteck des Dokuments ungültig ist und neu hergestellt werden muß. *EvOcViewPartInvalid* übergibt immer **true**.

EvOcViewSavePart

```
bool EvOcViewSavePart(TOcSaveLoad far& ocSave);
```

Befiehlt dem Server, die Daten des eingebetteten Objektes im Lager abzuspeichern. Um das Onjekt zu speichern, ruft *EvOcViewSavePart* das Objekt *TOleDocument* auf, das für jedes eingebettete Objekt das nötige Lager erzeugt. Speichert die Dimensionen der Serveransicht, die der Server dem Container angibt, damit dieser das eingebettete Objekt neu im Containerfenster zeichnen kann.

Siehe auch: *TOleView::EvOcViewClose*, *TOleView::EvOcViewOpenDoc*, *TOleDocument*

VnInvalidateRect

```
bool VnInvalidateRect(LPARAM p);
```

Macht den durch *p* angegebenen Ansichtsbereich umgültig. Verwenden Sie diese Funktion, um das abgrenzende Rechteck, das ein eingebettetes Objekt umgibt, wenn das Objekt - normalerweise durch direktes Bearbeiten verändert - wurde, ungültig zu machen. Wenn die Funktion erfolgreich war, übergibt sie *true*.

Einträge in die Anworttabelle

Einträge der Antworttabelle	Elementfunktion
EV_OC_VIEWOPENDOC	*EvOcViewOpenDoc*
EV_OC_VIEWINSMENUS	*EVOcViewInsMenus*
EV_OC_VIEWCLOSE	*EvOcViewClose*
EV_OC_VIEWSAVEPART	*EvOcViewSavePart*
EV_OC_VIEWLOADPART	*EvOcViewLoadPart*
EV_OC_VIEWATTACHWINDOW	*EvOcViewAttachWindow*

Klasse TOleWindow olewindo.h

Wird von *TWindow* abgeleitet. *TOleWindow* liefert Unterstützung, um Objekte in Verbunddokumenten einzubetten ud dient als Client eines Rahmenfensters. Ein Verbunddokument, wie das durch *TOleWindow* unterstützte, kann vom Arbeitsblatt bis hin zur Bitmap viele verschiedene Arten eingebetteter Objekte enthalten. Außer Unterstützung für eine Vielzahl von grundlegenden Fensteroperationen zu liefern, implementiert *TOleWindow* auch mehrere OLE-verwandte Operationen, unter ihnen:

- Antworten auf Ereignisse des Ziehens und Ablegens
- Direkte Bearbeitung (Der Prozess, währen dessen ein eingebttetes Objekt bearbeitet werden kann, ohne daß in die zugehörige Serveranwendung zugeschaltet werden muß)
- Aktivieren der Serveranwendung eines eingebetteten Objektes
- Erzeugen von Ansichten für die Containeranwendung
- Übertragen der Skalierungsinformation eines Dokuments zwischen einer Container- und einer Server-Ansicht

TOleWindow hat die Fähigkeit zu bestimmen, ob es als Server oder als Container agieren soll. Als Container besitzt *TOleWindow* einen Zeiger auf ein *TOcView*, und als Container richtet sie einen Zeiger auf ein *TOcRemView* ein. Aus Sicht des Servers besitzt jede Fernansicht ein korrespondierendes *TOleWindow*.

Durch seine vielen Elementfunktionen zur Ereignisbehandlung kommuniziert *TOleWindow* mit ObjectComponents, um Container- und Server-Unterstützung für eingebettete Objekte zu implementieren, Ansichten zu aktualisieren und auf eine Reihe von Menübefehlen zu antworten, die mit den typischen Befehlskennungen (z.B. CM_FILEMENU) assoziiert sind. Außerdem unterstützt sie spezifische OLE-Aktionsarten, wie die aus dem Edit-Menü aktivierten Arten (z.B. Edit und Open). Diese Befehle und Aktionsarten können aus verschiedenen Quellen stammen, wie z.B. einer Menüauswahl, einem Schaltfeld oder sogar aus einer internen Programm-Botschaft.

Umgekehrt kommuniziert ObjectComponents mit ObjectWindows mittels verschiedener EV_OC_Xxxx-Botschaften. Einige dieser Nachrichten, z.B. EV_OC_VIEWPARTINVALID, implementieren Container-Unterstützung, andere wiederum, z.B. EV_OC_VIEWCLOSE, Server-Unterstützung.

Für jede durch den Benutzer definierte, aus *TOleWindow* abgeleitete Klasse, muß eine Auswahl der geeigneten Funktionen getroffen werden. Wenn zusätzliche Server-Unterstützung geliefert werden soll, dann müssen nur die Funktionen, die Server-Nachrichten implementieren, definiert

werden; im Falle von Container-Unterstützung nur diejenigen Funktionen, die zusätzliche Container-Unterstützung liefern.

Die in der Container-Anwendung eingebetteten Daten (ein vollständiges Dokument mit einem oder mehreren eingebetteten Objekten) beispielsweise und die in der Server-Anwendung eingebetteten Daten (ein einzelnes OLE Objekt mit oder ohne anderen eingebetteten Objekten) können in die Lager geschrieben oder aus den Lagern gelesen werden. Wenn *TOleWindow* ohne *TOleView* verwendet wird, dann muß das Lager manipuliert werden, indem direkt mit der ObjectComponents-Klasse *TOcDocument* kommuniziert wird.

TOleWindow unterstützt außer der Kommunikation mit ObjectComponents-Klassen viele Transaktionen, die aus Interaktionen mit anderen ObjectComponents-Klassen resultieren. Da *TOleWindow* aus *TWindow* abgeleitet wurde, erbt es natürlich viele Funktionalitäten von *TWindow*.

Siehe auch: *TOcView*, *TWindowView*, *TWindow*

Öffentlicher Konstruktor und Destruktor

Konstruktor

```
Constructor TOleWindow(TWindow* parent = 0, TModule* module = 0);
```

Konstruiert ein Objekt *TOleWindow*, das mit dem spezifizierten übergeordneten Fenster und der Modulinstanz verknüpft ist.

Destruktor

```
~TOleWindow();
```

Überprüft, ob geöffnete Ansichten vorhanden sind, und, wenn keine offenen Ansichten existieren, zerstört das Objekt *TOleWindow*.

Öffentliche Elementfunktionen

```
virtual bool Deactivate();
```

Wenn ein eingebettetes Objekt nicht länger das aktive eingebettete Objekt ist, entweder weil der Benutzer die direkte Bearbeitungssitzung abgeschlossen hat oder weil der Benutzer außerhalb des eingebetteten Objekts angeklick hat, wird *Deactivate* aufgerufen, um das Objekt abzuwählen. *Deactivate* übergibt **true**, wenn sie erfolgreich war.

GetOcApp

```
TOcApp* GetOcApp();
```

Übergibt die ObjectComponents-Anwendung, die mit diesem Fenster verknüpft ist. Jede ObjectComponents-Anwendung, die Verknüpfen und Einbetten unterstützt, besitzt ein verknüpftes Objekt *TOcApp*.

Siehe auch: *TOcApp*

GetOcDoc

```
TOcDocument* GetOcDoc();
```

Übergibt das mit diesem Fenster verknüpfte ObjectComponents-Dokument. Dieses Dokument kann entweder ein Container- oder ein Server-Dokument sein. Wenn es ein durch den Container erzeugtes *TOcDocument* ist, dann ist das Dokument ein ganzes, vollständiges Dokument, das aus einem oder mehreren eingebetteten Objekten bestehen kann. Ist es ein durch den Server erzeugtes *TOcDocument*, dann besteht das Dokument aus den Daten eines einzelnen OLE Objekts.

Siehe auch: *TOcDocument*

GetOcRemView

```
TOcRemView* GetOcRemView();
```

Übergibt die mit diesem Fenster verknüpfte Ansicht des Servers. Um das OLE-Objekt im Containerfenster zeichnen zu können, erzeugt der Server eine Remote-View.

Siehe auch: *TOcRemView*

GetOcView

```
TOcView* GetOcView();
```

Zeigt auf den mit diesem Fenster verknüpften ObjectComponents-Container. Die Container-Ansicht enthält das vollständige Dokument (i.e. ein Dokument, das ein oder mehrere eingebettete Objekte enthält).

Siehe auch: *TOcView*

HasActivePart

```
bool HasActivePart();
```

Übergibt **true**, wenn die Ansicht des Containers ein aktives eingebettetes In-Place-Objekt enthält.

Siehe auch: *TOcView::GetActivePart*, *InvalidatePart*

PaintMetafile

```
virtual void PaintMetafile(TDC& dc, bool erase, TRect& rect);
```

Zeichnet den Bereich des Servers, in dem das eingettete Objekt liegt, neu. Der Parameter *dc* zeigt auf den Gerätekontext (DC). Wenn der Hintergrund des eingebetteten Objekts neu gezeichnet werden muß, dann ist der Parameter *erase* **true**. *rect* bezieht sich auf den neu zu zeichnenden Bereich.

Siehe auch: *InvalidatePart*

Geschützte Datenelemente

ContainerName

```
string ContainerName;
```

Enthält den Namen des Containers. Der Server stellt den Containernamen dar, wenn ein eingebettetes Objekt im Serverfenster editiert wird (auch *out-of-place* Editierung genannt).

DragDC

```
TDC* DragDC;
```

Zeigt auf das DC, das benutzt wird, wenn ein Objekt gezogen wird.

Siehe auch: *DragPt*

DragHit

```
TUIHandle::TWhere DragHit;
```

Zeigt die Position im eingebetteten Objekt an, an der der Benutzer die Maus angeklickt hat. Diese kann eine der Werte von *TUIHandle::TWhere* annehmen, z.B. *TopLeft*, *TopCenter*, *TopRight*, *MidLeft*, *MidCenter*, *MidRight*, *BottomLeft*, *BottomCenter*, *BottomRight* oder *Outside*, wenn nicht gezogen wird.

Siehe auch: *TUIHandle::TWhere enum*

DragPart

```
TOcPart* DragPart;
```

Zeigt auf das eingebettete Objekt (*Part*), das gezogen wird.

DragPt

```
TPoint DragPt;
```

Gibt den Punkt (in logischen Einheiten) an, wo sich der Mauszeiger über dem gezogenen Objekt befindet.

Siehe auch: *DragDC*

DragRect

```
TRect DragRect;
```

Enthält das gezogene Rechteck.

Siehe auch: *DragDC*

DragStart

```
TPoint DragStart;
```

Enthält den Ausgangspunkt, von dem das Ziehen des Objektes ausging.

Siehe auch: *DragPt*

Embedded

```
bool Embedded;
```

Ist **true**, wenn dieses *TOleWindow* ein eingebettetes Fenster ist. Daraus resultierend wird ein *TocRemView* erzeugt, da die Daten in einem Serverfenster dargestellt werden.

OcApp

```
TOcApp* OcApp;
```

Enthält die mit diesem *TOleWindow* verknüpfte ObjectComponents-Anwendung.

Siehe auch: *TOcView::OcApp*, *TOcApp*

OcDoc

```
TOcDocument* OcDoc;
```

Enthält das mit diesem *TOleWindow* verknüpfte ObjectComponents-Dokument.

Siehe auch: *TOcDocument*, *TOcView::OcDocument*

OcView

```
TOcView* OcView;
```

Enthält die mit dieserAnsicht von *TOleWindow* verknüpfte ObjectComponents-Ansicht oder Remote-View.

Siehe auch: *TOcView*

Pos

```
TRect Pos;
```

Enthält den aktuellen Bereich des Fensters, in dem das Objekt eingebettet ist. *Pos* spiegelt den Bereich wieder, in den das Objekt bewegt wird, sobald eine Objektbewegung stattfindet.

Scale

```
TOcScaleFactor Scale;
```

Enthält den aktuellen Skalierungsfaktor. Der Server benutzt diese Information, um zu bestimmen, wie das Dokument skaliert wird.

Siehe auch: *TOcScaleFactor*, *EvOcViewGetScale*, *SetScale*

TOleWindow::ShowObjects

```
bool ShowObjects;
```

Ist **true**, wenn der Rahmen des eingebetteten Objekts (die grauen oder schattierten Striche um das Objekt) dargestellt wird. Der Rahmen kann an- oder ausgestellt werden, je nachdem, ob das Objekt erscheinen soll oder nicht.

Geschützte Elementfunktionen

CanClose

```
bool CanClose();
```

Übergibt **true**, wenn das Fenster geschlossen werden kann. Überprüft alle Funktionen von *CanClose* der untergeordneten Fenster des Servers, die alle **true** wiedergeben müssen, damit das Fenster geschlossen werden kann. Beendet alle offenen Editiersitzungen, bevor sie das Fenster schließt. Ansonsten wird die Kontrolle an *TWindow::CanClose* weitergegeben.

CeEditConvert

```
void CeEditConvert(TCommandEnabler&);
```

Aktiviert einen Befehl mit der Kennung CM_EDITCONVERT, der den Benutzer das ausgewählte Objekt von einem Format in ein anderes konvertieren läßt. Dies ist eine OLE-spezifische Popup-Menüoption.

CeEditCopy

```
void CeEditCopy(TCommandEnabler&);
```

Aktiviert einen Befehl mit der Kennung CM_EDITCOPY, die den Benutzer ausgewählte Objekte in die Zwischenablage kopieren läßt.

CeEditCut

```
void CeEditCut(TCommandEnabler&);
```

Aktiviert einen Befehl mit der Kennung CM_EDITCUT, die den Benutzer ein ausgewähltes Objekt aus der Ansicht kopieren und löschen läßt.

CeEditDelete

```
void CeEditDelete(TCommandEnabler&);
```

Aktiviert einen Befehl mit der Kennung CM_EDITDELETE, die den Benutzer das ausgewählte Objekt aus der Ansicht löschen läßt.

CeEditInsertObject

```
void CeEditInsertObject(TCommandEnabler&);
```

Aktiviert einen Befehl mit der Kennung CM_EDITCUT, die es dem Benutzer erlaubt, einen Textausschnitt aus der Anssicht auszuschneiden.

CeEditLinks

```
void CeEditLinks(TCommandEnabler&);
```

Aktiviert einen Befehl mit der Kennung CM_EDITLINKS, die es dem Benutzer erlaubt, die Liste der verbundenen Einträge in der aktuellen Ansicht manuell zu aktualisieren.

CeEditObject

```
void CeEditObject(TCommandEnabler&);
```

Aktiviert einen Befehl mit der Kennung CM_EDITOBJECT, die es dem Benutzer erlaubt, das eingebettete Objekt zu editieren.

CeEditPaste

```
void CeEditPaste(TCommandEnabler&);
```

Aktiviert einen Befehl mit der Kennung CM_EDITPASTE, die es dem Benutzer erlaubt, das eingebettete Objekt aus der Zwischenablage einzufügen.

CeEditPasteLink

```
void CeEditPasteLink(TCommandEnabler&);
```

Aktiviert einen Befehl mit der Kennung CM_EDITPASTELINK, die es dem Benutzer erlaubt, ein eingebettetes Objekt in der Zwischenablage zu verbinden. S. die Konstante ocrxxxx der Zwischenablage für eine Beschreibung der vorhandenen Formate der Zwischenablage.

Siehe auch: *TOleWindow::CeEditPasteSpecial*, *ocrxxxx Konstanten der Zwischenablage*

CeEditPasteSpecial

```
void CeEditPasteSpecial(TCommandEnabler&);
```

Aktiviert den Befehl *PasteSpecial*, der es dem Benutzer erlaubt, ein Format der Zwischenablage auszuwählen, damit dieses eingefügt oder verbunden eingefügt werden kann. S. die Konstante ocrxxxx der Zwischenablage für eine Beschreibung der vorhandenen Formate der Zwischenablage.

Siehe auch: *TOleWindow::CeEditPasteLink*, *ocrxxxx Konstanten der Zwischenablage*

CeEditVerbs

```
void CeEditVerbs(TCommandEnabler& ce);
```

Aktiviert den Befehl Bearbeiten/Aktionsarten, der es dem Benutzer erlaubt, eines der spezifischen OLE-Aktionsarten aus dem Edit-Menü auszuwählen, wie z.B. Edit, Open oder Play.

CeFileClose

```
void CeFileClose(TCommandEnabler& ce);
```

Aktiviert den Befehl FileClose, der es dem Benutzer erlaubt, das Ansichtsfenster zu verlassen.

CleanupWindow

```
void CleanupWindow();
```

Führt die normale Fensterbereinigung durch und informiert das Objekt *TOcView*, das das Fenster geschlossen ist.

Siehe auch: *TWindow::CleanupWindow*, *TOcView*

CmEditConvert

```
void CmEditConvert();
```

Antwortet auf einen Befehl mit der Kennung CM_EDITCONVERT, durch Konvertierung eines Objektes von einem Typ in einen anderen.

CmEditCopy

```
void CmEditCopy();
```

Antwortet auf einen Befehl mit der Kennung CM_EDITCOPY, durch Kopieren des ausgewählten Textes in die Zwischenablage.

CmEditCut

```
void CmEditCut();
```

Antwortet auf einen Befehl mit der Kennung CM_EDITCUT durch kopieren des ausgewählten Textes in die Zwischenablage, before der Text ausgeschnitten wird.

CmEditDelete

```
void CmEditDelete();
```

Antwortet auf einen Befehl mit der Kennung CM_EDITDELETE durch Löschen des ausgewählten Textes.

CmEditInsertObject

```
void CmEditInsertObject();
```

Antwortet auf einen Befehl mit der Kennung CM_EDITINSERTOBJECT durch Einfügen eines Objektes, das der Benutzer aus einer Liste von Objekttypen auswählt.

CmEditLinks

```
void CmEditLinks();
```

Antwortet auf einen Befehl mit der Kennung CM_EDITLINKS durch Aktualisieren der vom Benutzer ausgewählten Liste von in der aktuellen Ansicht verbundenen Einträgen.

CmEditPaste

```
void CmEditPaste();
```

Antwortet auf einen Befehl mit der Kennung CM_EDITPASTE durch Einfügen eines Objektes aus der Zwischenablage in das Dokument.

CmEditPasteLink

```
void CmEditPasteLink();
```

Antwortet auf einen Befehl mit der Kennung CM_EDITPASTELINK durch Erzeugen einer Verbindung zwischen dem aktuellen Dokument und dem Objekt in der Zwischenablage.

Siehe auch: *TOleWindow::CmEditPasteSpecial*, *ocrxxxx Clipboard-Konstanten*

CmEditPasteSpecial

```
void CmEditPasteSpecial();
```

Antwortet auf einen Befehl mit der Kennung CM_EDITPASTESPECIAL, indem es dem Benutzer erlaubt, ein Objekt aus einer Liste mit vorkom-

menden Formaten auszuwählen, damit aus der Zwischenablage in das Dokument eingefügt werden kann.

Siehe auch: *TOleWindow::CmEditPasteLink*, *ocrxxxx Clipboard-Konstanten*

CmFileClose

```
void CmFileClose();
```

Antwortet auf einen Befehl mit der Kennung CM_FILECLOSE, indem es die Nachricht WM_CLOSE an das übergeordnete Fenster schickt, damit es die Anwendung schließt.

CreateOcView

```
virtual TOcView* CreateOcView(TDocTemplate* tpl, bool isEmbedded,
IUnknown* outer);
```

Erzeugt eine mit dem eingebetteten Objekt verknüpfte ObjectComponents-Ansicht. Verknüpft die Ansicht mit der in *tpl* spezifizierten Dokumenten-Template. Wenn *isEmbedded* **true** ist, dann wird eine Fernansicht erzeugt (d.h. ein *TOcRemView*, anstelle eines *TOcView*). Der Parameter *outer* bezieht sich auf die Schnittstelle *IUnknown*, mit dem sich die Ansicht zusammenschließt.

Siehe auch: *TOcView*, *TOcRemView*

CreateVerbPopup

```
TPopupMenu* CreateVerbPopup(const TOcVerb& ocVerb);
```

Erzeugt und Aktiviert eine Popup-Menüoption (*ocVerb*) des Editiermenüs. Die Aktionsart überschreibt eine Aktion (z.B. Edit, Open, Play), die für das eingebettete Objekt geeignet ist.

Siehe auch: *EvDoVerb*

EvCommand

```
LRESULT EvCommand(uint id, HWND hWndCtl, uint notifyCode);
```

Überschreibt die übliche Nachricht *EvCommand*, um die OLE-Aktionsarten aus CM_EDITFIRSTVERB an CM_EDITLASTVERB weiterzugeben. Diese Befehle, die in oleview.rh definiert werden, korrespondieren mit den spezifischen OLE-Editmenüauswahlpunkten wie z.B. Edit, Open und Play. Alle anderen Befehle werden an *TWindow::EvCommand* zur normalen Verarbeitung weitergegeben.

Siehe auch: *CM_xxxx Edit View Konstanten*, *TOleWindow::EvCommandEnable*, *TWindow::EvCommand*

EvCommandEnable

```
void EvCommandEnable(TCommandEnabler& commandEnabler);
```

Überschreibt die übliche Nachricht *EvCommandEnable*, um die OLE-Aktionsarten aus CM_EDITFIRSTVERB an CM_EDITLASTVERB weiterzugeben. Diese Befehle aktivieren die spezifischen OLE-Editmenüauswahlpunkten wie z.B. Edit, Open und Play. Alle anderen Befehle werden an *TWindow::EvCommand* zur normalen Verarbeitung weitergegeben.

Siehe auch: *TOleWindow::EvCommand*, *TWindow::EvCommand*

EvDoVerb

```
void EvDoVerb(uint whichVerb);
```

Führt eine OLE-verwandte Menüoption , die mit dem ausgewählten Objekt assoziiert ist, aus dem Editmenü aus (z.B. Edit, Copy oder Play).

Siehe auch: *CreateVerbPopup*

EvLButtonDblClk

```
void EvLButtonDblClk(uint modKeys, TPoint& point);
```

Antwortet auf eine durch einen Doppelklick der Maustasten gesandte Nachricht. *EvLButtonDblClk* überprüft, welches eingebettete Objekt angeklickt wurde und aktiviert dann das eingebettete Objekt an Ort und Stelle.

EvLButtonDown

```
void EvLButtonDown(uint modkeys, TPoint& point);
```

Antwortet auf eine durch einen linken Maustastenklick gesandte Nachricht, indem sie eine Maus-Drag Transaktion am gegebenen Punkt beginnt. Überprüft, ob eines der eingebetteten Objekte angeklickt wurde. Der Parameter *modKeys* enthält die Werte für Tastenkombinationen, wie z.B. Shift und Maustasten-Doppelklick.

Siehe auch: *EvRButtonDown*

EvLButtonUp

```
void EvLButtonUp(uint modKeys, TPoint& point);
```

Antwortet auf ein Loslassen der linken Maustaste, indem sie die Maus-Drag Aktion beendet. Der Parameter *point* bezieht sich auf die Stelle, an der sich der Mauszeiger befindet. *modKeys* enthält die Werte kombinerte Tasten und Maus Transaktionen.

EvMDIActivate

```
void EvMDIActivate(HWND hWndActivated, HWND hWndDeactivated);
```

Antwortet auf eine vom untergeordneten MDI Fenster kommende Nachricht (wenn eine existiert) und läßt die TOcView Klasse wissen, daß das untergeordnete Fenster des Ansichtsfensters aktiviert oder deaktiviert wurde.

Der Parameter *hWndActivated* enthält eine Kennung zu dem untergeordneten MDI-Fenster, das aktiviert wird. Sowohl Aktivieren als auch Deaktivieren (*hWndDeactivated*) des untergeordneten Fensters erhalten diese Nachricht.

Siehe auch: *TOcView*

EvMouseMove

```
void EvMouseMove(uint, TPoint& point);
```

Antwortet auf eine Bewegung mit der Maus mit der geeigneten Transaktion. Wenn die Maus gezogen wird, wird das eingebettete Objekt bewegt. Wenn eine Größen verändernde Operation vorgenommen wird, wird die Größe des eingebetteten Objekts verändert. Diese Nachricht wird nur geschickt, wenn eine der beiden genannten Aktionen das eingebette Objekt betrifft.

EvOcEvent

```
LRESULT EvOcEvent(WPARAM wParam, LPARAM lParam);
```

Antwortet auf die Nachricht WM_OCEVENT und gibt diese Nachricht basierend auf dem Parameter *wParam* weiter. ObjectComponents sendet WM_OCEVENT-Nachrichten, wenn es mit einem mit OLE generierten Ereignis kommunizieren muß; z.B., wenn ein Server Mauspaletten darstellen soll.

EvOcPartInvalid

```
bool EvOcPartInvalid(TOcPart far&);
```

Verwaltet eine WM_OCEVENT Nachricht, die das verbundene oder eingebettete Objekt im Dokument betrifft un erklärt der Teil (Part) für ungültig.

Wenn das *TOleWindow* Objekt nicht in der Lage ist die Nachricht zu bearbeiten, dann übergibt *EvOcPartInvalid* false.

EvOcViewAttachWindow

```
bool EvOcViewAttachWindow(bool attach);
```

Verknüpft die Ansicht mit dem übergeordneten ObjectWindows Fenster, damit der Benutzer das eingebettete Objekt offen edititeren kann, oder wenn das eingebettete Objekt während einer In-Place Editiersitzung deaktiviert wurde.

Wenn das *TOleWindow* Objekt nicht in der Lage ist die Nachricht zu bearbeiten, dann übergibt *EvOcViewAttachWindow* **false**.

EvOcViewBorderSpaceReq

```
bool EvOcViewBorderSpaceReq(TRect far*);
```

Verlangt, daß der Server Platz für eine Mauspalette in der Ansicht eines eingebetteten Objektes erzeugt.

Wenn das *TOleWindow* Objekt nicht in der Lage ist die Nachricht zu bearbeiten, dann übergibt *EvOcViewBorderSpaceReq* **false**, was auch der voreingestellte Wert ist.

Siehe auch: *EvOcViewBorderSpaceSet*

EvOcViewBorderSpaceSet

```
bool EvOcViewBorderSpaceSet(TRect far*);
```

Verlangt, daß sie Mauspalette eines Servers in der Containeransicht eines eingebetteten Objekts plaziert wird.

Wenn das *TOleWindow* Objekt nicht in der Lage ist die Nachricht zu bearbeiten, dann übergibt *EvOcViewBorderSpaceSet* **false**, was auch der voreingestellte Wert ist.

Siehe auch: *EvOcViewBorderSpaceReq*

EvOcViewClipData

```
HANDLE EvOcViewClipData(TOcFormat far&);
```

Verlangt Daten der Zwischenablage im spezifizierten Format.

Wenn das *TOleWindow* Objekt nicht in der Lage ist die Nachricht zu bearbeiten, dann übergibt *EvOcViewClipData* **false**.

EvOcViewClose

```
bool EvOcViewClose();
```

Befiehlt dem Server ein zur Zeit offenes Dokument und die verknüpfte Ansicht zu schließen.

Wenn das *TOleWindow* Objekt nicht in der Lage ist die Nachricht zu bearbeiten, dann übergibt *EvOcViewClose* **false**.

EvOcViewDrag

```
bool EvOcViewDrag(TOcDragDrop far&);
```

Bearbeitet eine OC_VIEWDRAG-Nachricht, indem sie dem Container befiehlt, visuelles Feedback zu liefern, während der Benutzer das eingebettete Objekt zieht.

Wenn das *TOleWindow* Objekt nicht in der Lage ist die Nachricht zu bearbeiten, dann übergibt *EvOcViewDrag* **false**.

EvOcViewDrop

```
bool EvOcViewDrop(TOcDragDrop far&);
```

Verlangt, das ein gegebenes Objekt an einer bestimmten Stelle im Containerfenster fallengelassen wird.

Wenn das *TOleWindow* Objekt nicht in der Lage ist die Nachricht zu bearbeiten, dann übergibt *EvOcViewDrop* **false**.

EvOcViewGetPalette

```
bool EvOcViewGetPalette(LOGPALETTE far* far* palette);
```

Verlangt nach der Farbpalette, um das Objekt zu zeichnen.

Wenn das *TOleWindow* Objekt nicht in der Lage ist die Nachricht zu bearbeiten, dann übergibt *EvOcViewGetPalette* false.

EvOcViewGetScale

```
bool EvOcViewGetScale(TOcScaleFactor& scaleFactor);
```

Antwortet auf eine OC_VIEWGETSCALE Nachricht und holt die Skalierung für das Server Objekt, damit das eingebettete Objekt im richtigen Skalierungswert (z.B. 120%) dargestellt wird. *scaleFactor* zeigt den Skalierungsfaktor an, i.e. die Proportion zwischen der Größe des eingebetteten Objekts und der Größe der Stelle, an der das Objekt dargestellt werden soll.

Wenn das *TOleWindow* Objekt nicht in der Lage ist die Nachricht zu bearbeiten, dann übergibt *EvOcViewGetScale* false.

Siehe auch: *Scale*, *TOcScaleFactor*

EvOcViewGetSiteRect

```
bool EvOcViewGetSiteRect(TRect far* rect);
```

Holt die Größe des Rechtecks, in dem des eingebettete Objekt plaziert werden soll. *rect* bezieht sich auf die Größe des abgrenzenden Rechtecks, das das Objekt umschließt.

Siehe auch: *EvOcViewSetSiteRect*

EvOcViewInsMenus

```
bool EvOcViewInsMenus(TOcMenuDescr far&);
```

Verlangt, das die Menüs sich in einem zusammengesetzten Menü (ein Menü, das aus Server- und Container-Menüs zusammengesetzt wird) befinden.

Wenn das *TOleWindow* Objekt nicht in der Lage, ist die Nachricht zu bearbeiten, übergibt *EvOcViewInsMenu* **false**.

EvOcViewLoadPart

```
bool EvOcViewLoadPart(TOcSaveLoad far* ocLoad);
```

Verlangt, daß ein eingebettetes Objekt sich selbst lädt.

Wenn das *TOleWindow* Objekt nicht in der Lage ist die Nachricht zu bearbeiten, dann übergibt *EvOcViewLoadPart* **false**.

EvOcViewOpenDoc

```
bool EvOcViewOpenDoc(const char far*);
```

Befiehlt dem Container ein existierendes Dokument zu öffnen, das zum Verbinden von der Einbettungsstelle benutzt wird.

Wenn das *TOleWindow* Objekt nicht in der Lage ist die Nachricht zu bearbeiten, dann übergibt *EvOcViewOpen Doc* **false**.

EvOcViewPaint

```
bool EvOcViewPaint(TOcViewPaint far&);
```

Befiehlt dem Server ein Objekt an einer angegebenen Stelle im spezifizierten Gerätekontext zu zeichnen.

Wenn das *TOleWindow* Objekt nicht in der Lage ist die Nachricht zu bearbeiten, dann übergibt *EvOcViewPaint* **false**.

EvOcViewPartInvalid

```
bool EvOcViewPartInvalid(TOcPart far&);
```

Informiert einen aktiven Container, daß eines seiner eingebetteten Objekte neu gezeichnet werden muß. Änderungen im Teil (Part) des Containers sollten in jeder anderen, nicht aktiven Ansicht wiedergegeben werden. Übergibt true nachdem alle Ansichten von den notwendigen Änderungen in Kenntnis gesetzt worden sind.

Wenn das *TOleWindow* Objekt nicht in der Lage ist die Nachricht zu bearbeiten, dann übergibt *EvOcViewPartInvalid* **false**.

EvOcViewPartSize

```
bool EvOcViewPartSize(TRect far*);
```

Der Server fragt sich selbst nach der Größe seines aktuellen Rechtecks und läßt den Container wissen, wie groß die Server-Ansicht in Pixel ist.

Wenn das *TOleWindow* Objekt nicht in der Lage ist die Nachricht zu bearbeiten, dann übergibt *EvOcViewPartSize* **false**.

EvOcViewSavePart

```
bool EvOcViewSavePart(TOcSaveLoad far& ocSave);
```

Befiehlt dem Server Daten eines eingebetteten Objekts (das von dem Parameter *ocSave* wiedergegebene Teil) in das Lager zu schreiben.

Wenn das *TOleWindow* Objekt nicht in der Lage ist die Nachricht zu bearbeiten, dann übergibt *EvOcViewSavePart* **false**.

EvOcViewScroll

```
bool EvOcViewScroll(TOcScrollDir);
```

Befiehlt dem Container das Ansichtsfenster zu durchlaufen und aktualisiert, wenn dies notwendig ist, jeden internen Status. *EvOcViewScroll* wird aufgerufen, wenn die Server-Größe verändert wird oder eine Drop-Aktion in der Nähe des Fensters stattfindet.

Wenn das *TOleWindow* Objekt nicht in der Lage ist die Nachricht zu bearbeiten, dann übergibt *EvOcViewScroll* **false**.

EvOcViewSetScale

```
bool EvOcViewSetScale(TOcScaleFactor& scaleFactor);
```

Antwortet auf eine OC_VIEWSETSCALE Nachricht und bearbeitet die Skalierung für Server-Anwendungen. Sie stellt dabei sicher, daß das eingebettete Objekt mit den richtigen Skalierungswerten dargestellt wird (z.B. 120%). Der Server diesen Wert in seiner Zeichenprozedur, wenn das eingebettete Objekt neu gezeichnet werden muß. *scaleFactor* zeigt den Skalierungsfaktor an, i.e. die Proportion zwischen der Größe des eingebetteten Objekts und der Größe der Stelle, an der das Objekt dargestellt werden soll.

Wenn das *TOleWindow* Objekt nicht in der Lage ist die Nachricht zu bearbeiten, dann übergibt *EvOcViewSetScale* false.

Siehe auch: *EvOcViewGetScale*, *TOcScaleFactor*

EvOcViewSetSiteRect

```
bool EvOcViewSetSiteRect(TRect far* rect);
```

Konvetiert *rect* in logische Einheiten. Dieser Bereich (*Site*) wird in logischen Einheiten gemessen, die jeden Skalierungsfaktor berücksichtigen. *rect* bezieht sich auf die Größe des abgrenzenden Rechtecks, das das eingebettete Objekt umschließt.

Siehe auch: *EvOcViewGetSiteRect*

EvOcViewShowTools

```
bool EvOcViewShowTools(TOcToolBarInfo far& tbi);
```

Befiehlt dem Server, seine Mauspaletten zur Darstellung im Container-Fenster zur Verfügung zu stellen. Übergibt **true**, wenn die Mauspaletten geliefert wurden.

Wenn das *TOleWindow* Objekt nicht in der Lage ist die Nachricht zu bearbeiten, dann übergibt *EvOcViewShowTools* **false**.

EvOcViewTitle

```
const char far* EvOcViewTitle();
```

ragt den Container nach dem Titel seines Rahmenfensters und übergibt diesen Titel.

EvRButtonDown

```
void EvRButtonDown(uint modKeys, TPoint& point);
```

Antwortet auf drücken der rechten Maustaste. Überprüft, welches eingebettete Objekt angeklickt wird und stellt ein lokales Menü mit geeigneten Optionen für das eingebettete Objekt dar.

point bezieht sich auf die Stelle, an der sich der Mauszeiger befindet. *modKeys* enthält die Werte für eine kombinierte Tasten und Maus Transaktion wie z.B. Shift und Maustasten-Doppelklick.

Siehe auch: *EvLButtonDown*

EvSetCursor

```
bool EvSetCursor(HWND, uint hitTest, uint);
```

Überprüft, an welcher Stelle im Fenster sich der Cursor befindet und über welches Objekt sich der Cursor bewegt. Wenn der Cursor sich in einem eingebetteten Objekt befindet, dann ändert *EvSetCursor* das Aussehen des Cursors.

Wenn sich der Server über einem inaktiven Teil und nicht über einem Griff befindet, dann benutzt *EvSetCursor* einen Pfeil. Befindet sich der Cursor über einem Griff des eingebetteten Teils, dann wird der Cursor zum Größen verändernden Cursor.

EvSetFocus

```
void EvSetFocus(HWND hWndLostFocus);
```

Antwortet auf eine Änderung im Fokus des Fensters. *hWndLostFocus* enthält eine Kennung des Fensters, das den Fokus verliert. *EvSetFocus* prüft, ob ein In-Place Server existiert und, wenn ja, übegibt des Fokus an den In-Place Server.

EvSize

```
void EvSize(uint sizeType, TSize& size);
```

Übergibt das Ereignis an *TWindow::EvSize*, damit es normal verarbeitet wird, und läßt einen möglichen In-Place Server seine Größe anpassen.

Siehe auch: *TOcView::EvSize*

GetInsertPosition

```
virtual void GetInsertPosition(TRect& rect);
```

Holt die Stelle (*rect*), an der das eingebettete Objekt eingefügt werden soll. Diese Funktion muß überschrieben werden, wenn eine Voreinstellung überschrieben werden soll.

GetLogPerUnit

```
virtual void GetLogPerUnit(TSize& logPerUnit);
```

Holt die logischen Einheiten (normalerweise Pixel pro Inch) eines Dokuments, damit die eingebetteten Objekte des Dokuments korrekt auf der Bildschirm DC gezeichnet werden können.

Init

```
void Init();
```

Initialisiert das Objekt *TOleWindow* mit dem geeigneten Fensterstil und initialisiert die nötigen Datenelemente (z.B. setzt die Beschleuniger ID auf IDA_OLEVIEW).

InvalidatePart

```
virtual void InvalidatePart(TOcInvalidate invalid);
```

Erklärt den Bereich, in dem sich das eingebettete Objekt exisitert, für ungültig. Der Server verwendet diese Funktion, um OLE mitzuteilen, daß das Teil (eingebettete Objekt) sich verändert hat. OLE befiehlt dann dem Server das Teil in ein neues Metafile zu zeichnen, damit OLE das Objekt für die Container-Anwendung sogar dann neu zeichnen kann, wenn die Server-Anwendung nicht aktiv ist.

Siehe auch: *PaintMetafile*, *TOcView_GetActivePart*

Paint

```
void Paint(TDC& dc, bool erase, TRect& rect);
```

Zeichnet den Fensterinhalt neu und teilt jedem eingebetteten Objekt mit, sich selbst neu zu zeichnen.

dc zeigt zum Pain DC. *erase* zeigt an, ob der Hintergrund gelöscht werden soll. *rect* ist eine Referenz zum abgrenzenden Rechteck des neu zu zeichnenden Bereichs.

PaintParts

```
virtual bool PaintParts(TDC& dc, bool erase, TRect& rect, bool metafile);
```

Zeichnet das eingebettete Objekt auf dem gegebenen DC neu.

der Parameter *erase* ist true, wenn der Hintergrund des eingebetteten Objekts neu gezeichnet werden soll. *rect* zeigt an, daß der Bereich neu gezeichnet werden muß. *metafile* an, ob der DC ein Metafile darstellt oder nicht.

Select

```
virtual bool Select(uint modKeys, TPoint& point);
```

Wählt das eingebettete Objekt am spezifizierten Punkt aus (gemessen in logischen Einheiten).

Übergibt true, wenn das Objekt durch den Mauszug eingefangen wurde. Ansonsten wird false übergeben.

SelectEmbedded

```
bool SelectEmbedded();
```

Wählt das eingebettete Objekt aus und übergibt **true**, um anzuzeigen, daß das Objekt ausgewählt wurde.

SetScale

```
virtual void SetScale(uint16 percent);
```

Setzt die Proportionen des eingebetteten Objekts zur Größe der Stelle fest.

Siehe auch: *SetupDC*

SetSelection

```
void SetSelection(TOcPart* part);
```

Wählt das eingebettete Objekt, das durch den Parameter *part* angegeben wird, aus. Wenn das eingebettete Objekt ausgwählt wird, dann wird ein Auswahlrechteck um den Bereich des Objektes gezeichnet. Nachdem ein eingebettetes Objekt ausgewählt wurde, kann den Benutzer Operationen

für das Objekt ausführen: z.B. bewegen, Größe verändern oder das Objekt in die Zwischenablage kopieren.

SetupDC

```
virtual void SetupDC(TDC& dc, bool scale = true);
```

Bestimmt den Ursprung und das Ausmaß (die logischen Koordinaten und die Größe des DC) des Viewports. Setzt den Greätekontext (DC) bevor sie das eingebettete Objekt zeichnet. *dc* bezieht sich auf den DC und *scale* zeigt an, daß der beim Zeichnen des eingebetteten Objektes zu benutzende Skalierungsfaktor eine Bruch, der sich aus der Stelle und dem eingebetteten Objekt berechnet, ist.

Siehe auch: *TOcScaleFactor*, *SetScale*

SetupWindow

```
void SetupWindow();
```

Stellt eine Verbindung zwischen dem *TOcView* Objekt und dem HWND der Ansicht her, damit die Ansicht Benachrichtigungen an das Fenter schicken kann.

VnInvalidateRect

```
bool VnInvalidateRect(LPARAM p);
```

Wenn das eingebettete Objekt modifiziert wird, dann schickt *VnInvalidateRect* eine Nachricht an den View Teil des Doc/View Paares. Wenn *TOleWindow* diese Nachricht erhält, dann wird der Ansichtsbereich markiert, um gelöscht zu werden. *VnInvalidateRect* übergibt immer **true**.

Einträge in die Anworttabelle

Einträge der Antworttabelle	Elementfunktion
EV_WM_LBUTTONDOWN	EvLButtonDown
EV_WM_RBUTTONDOWN	EvRButtonDown
EV_WM_LBUTTONDBLCLK	EvLButtonDblClk
EV_WM_MOUSEMOVE	EvMouseMove
EV_WM_LBUTTONUP	EvLButtonUp
EV_WM_SIZE	EvSize
EV_WM_MDIACTIVATE	EvMDIActivate
EV_WM_SETFOCUS	EvSetFocus
EV_WM_SETCURSOR	EvSetCursor
EV_COMMAND(CM_EDITDELETE, CmEditDelete)	CmEditDelete
EV_COMMAND_ENABLE(CM_EDITDELETE, CeEditDelete)	CeEditDelete
EV_COMMAND(CM_EDITCUT, CmEditCut)	CmEditCut
EV_COMMAND_ENABLE(CM_EDITCUT, CeEditCut)	CeEditCut
EV_COMMAND(CM_EDITCOPY, CmEditCopy)	CmEditCopy
EV_COMMAND_ENABLE(CM_EDITCOPY, CeEditCopy)	CeEditCopy
EV_COMMAND(CM_EDITPASTE, CmEditPaste)	CmEditPaste
EV_COMMAND_ENABLE(CM_EDITPASTE, CeEditPaste)	CeEditPaste
EV_COMMAND(CM_EDITPASTESPECIAL, CmEditPasteSpecial)	CmEditPasteSpecial
EV_COMMAND_ENABLE(CM_EDITPASTESPECIAL, CeEditPasteSpecial)	CeEditPasteSpecial

EV_COMMAND(CM_EDITPASTELINK, CmEditPasteLink)	CmEditPasteLink
EV_COMMAND_ENABLE(CM_EDITPASTELINK, CeEditPasteLink)	CeEditPasteLink
EV_COMMAND(CM_EDITINSERTOBJECT, CmEditInsertObject)	CmEditInsertObject
EV_COMMAND_ENABLE(CM_EDITINSERTOBJECT, CeEditInsertObject)	CeEditInsertObject
EV_COMMAND_ENABLE(CM_EDITLINKS, CeEditLinks)	CeEditLinks
EV_COMMAND(CM_EDITLINKS, CmEditLinks)	CmEditLinks
EV_COMMAND_ENABLE(CM_EDITOBJECT, CeEditObject)	CeEditObject
EV_COMMAND_ENABLE(CM_EDITCONVERT, CeEditConvert)	CeEditConvert
EV_COMMAND(CM_EDITCONVERT, CmEditConvert)	CmEditConvert
EV_COMMAND(CM_EXIT, CmFileClose)	CmFileClose
EV_COMMAND_ENABLE(CM_EXIT, CeFileClose)	CeFileClose
EV_MESSAGE(WM_OCEVENT, EvOcEvent)	EvOcEvent
EV_OWLNOTIFY(vnInvalidate)	VnInvalidateRect
EV_OC_VIEWPARTINVALID	EvOcViewPartInvalid
EV_OC_VIEWTITLE	EvOcViewTitle
EV_OC_VIEWBORDERSPACEREQ	EvOcViewBorderSpaceReq
EV_OC_VIEWBORDERSPACESET	EvOcViewBorderSpaceSet
EV_OC_VIEWDROP	EvOcViewDrop
EV_OC_VIEWDRAG	EvOcViewDrag
EV_OC_VIEWSCROLL	EvOcViewScroll
EV_OC_VIEWGETSCALE	EvOcViewGetScale
EV_OC_VIEWGETSITERECT	EvOcViewGetSiteRect
EV_OC_VIEWSETSITERECT	EvOcViewSetSiteRect
EV_OC_VIEWPAINT	EvOcViewPaint
EV_OC_VIEWSAVEPART	EvOcViewSavePart
EV_OC_VIEWLOADPART	EvOcViewLoadPart
EV_OC_VIEWINSMENUS	EvOcViewInsMenus
EV_OC_VIEWSHOWTOOLS,	EvOcViewshowTools
EV_OC_VIEWGETPALETTE	EvOcViewGetPalette
EV_OC_VIEWCLIPDATA,	EvOcViewclipData
EV_OC_VIEWCLOSE	EvOcViewClose
EV_OC_VIEWPARTSIZE	EvOcViewPartSize
EV_OC_VIEWOPENDOC	EvOcViewOpenDoc
EV_OC_VIEWATTACHWINDOW	EvOcViewAttachWindow
EV_OC_VIEWSETSCALE	EvOcViewSetScale

Klasse TOpenSaveDialog opensave.h

TOpenSaveDialog ist die Basisklasse für modale Dialoge, die das Öffnen und Speichern von Dateien unter angegebenen Namen ermöglichen. *TOpenSaveDialog* konstruiert eine *TData*-Struktur und übergibt ihr den *TOpenSaveDialog*-Konstruktor. Dann wird der Dialog ausgeführt (modal) oder erzeugt (nicht-modal). Bei Rückgabe werden die notwendigen Felder aktualisiert, einschließlich eines Fehlerfeldes, das 0 enthält oder einen allgemeinen Dialogfehler.

Öffentlicher Konstruktor

Konstruktor

```
TOpenSaveDialog(TWindow* parent, TData& data, TResId templateID = 0,
                const char far* title = 0, TModule* module = 0);
```

Konstruiert ein Dialogfensterobjekt zum Öffnen und Sichern mit dem angegebenen übergeordneten Fenster, Daten, Ressourcen-ID, Titel und aktuellem Modulobjekt.

Siehe auch: *TOpenSaveDialog::TDataStruct*

Öffentliche Elementfunktionen

GetFileTitle

```
static int GetFileTitle(const char far* fileName,
                        char far* fileTitle, int fileTitleLen);
```

Speichert den Namen der Datei, die geöffnet oder gespeichert werden soll.

GetFileTitleLen

```
static int GetFileTitleLen(const char far* fileName);
```

Speichert die Länge des Namens der Datei, die geöffnet oder gespeichert werden soll.

Geschützte Datenelemente

Data

```
TData& Data;
```

Speichert den Dateinamen, seine Länge, Erweiterung, Filter, Ursprungsverzeichnis, die Standarderweiterung des Dateinamens und alle eventuellen Fehlermeldungen.

ofn

```
OPENFILENAME ofn;
```

Enthält die Attribute des Dateinamens, wie Länge, Erweiterung und Verzeichnis. *ofn* wird unter Verwendung der Felder in der *TOpenSaveDialog::TData*-Klasse initialisiert. Dieses Datenelement steht nicht unter Presentation Manager zur Verfügung.

Siehe auch: *Struktur TData*

ShareViMsgId

```
static uint shareViMsgId;
```

Enthält die ID der registrierten *ShareViolation*-Meldung. Dieses Datenelement steht nicht unter Presentation Manager zur Verfügung.

Siehe auch: Struktur *TData, ShareViolation*

Geschützte Konstruktoren

Konstruktor

```
TOpenSaveDialog(TWindow* parent, TData& data, TModule* module);
```

Konstruiert ein *TOpenSaveDialog*-Dialogfensterobjekt mit dem angegebenen übergeordneten Fensterobjekt, dem Daten- und dem aktuellen Modulobjekt.

Siehe auch: Struktur *TData struct*

Geschützte Elementfunktionen

CmLbSelChanged

```
void CmLbSelChanged();
```

Zeigt eine Änderung im Auswahlstatus des Dateinamen-Listenfensters in den Dialogfenstern *GetOpenFileName* oder *GetSaveFileName* an. *CmLbSelChanged* ist eine Standard-Behandlungsroutine für Befehlsbotschaften, die von *lst1* oder *lst2* (beziehungsweise von den Datei- und Verzeichnislistenfenstern) gesendet werden.

CmOk

```
void CmOk();
```

Reagiert auf Anklicken des OK-Feldes des Dialogfensters (mit dem Bezeichner IDOK). Ruft *CloseWindow* auf (und übergibt IDOK).

Siehe auch: *TDialog::CloseWindow*

DialogFunction

```
bool DialogFunction(uint message, WPARAM, LPARAM);
```

Gibt **true** zurück, wenn eine Botschaft behandelt wird, *ShareViMsgId*, wenn die Bedingung für gemeinsamen Dateizugriff verletzt wird (share violation), andernfalls **false**.

DoExecute

```
int DoExecute() = 0;
```

Erzeugt ein modales Dialogfenster und führt es aus.

Init

```
void Init(TResID templateID);
```

Initialisiert ein *TOpenSaveDialog*-Objekt mit der aktuellen Ressourcen-ID.

ShareViolation

```
virtual int ShareViolation();
```

Wenn beim Öffnen oder Speichern einer Datei eine Schutzverletzung (der Bedingung für gemeinsamen Dateizugriff) auftritt, wird *ShareViolation* aufgerufen, um eine Reaktion zu erhalten. Der Standard-Rückgabewert ist OFN_SHAREWARN. Andere Reaktionen auf Schutzverletzungen sind in dieser Tabelle aufgeführt. Dieses Datenelement steht nicht unter Presentation Manager zur Verfügung.

Konstante	Bedeutung
OFN_SHAREFALLTHROUGH	Gibt an, daß der Dateiname verwendet werden kann und das Dialogfenster ihn an die Anwendung zurückgeben soll.
OFN_OFN_SHARENOWARN	Weist das Dialogfenster an, keine weiteren Aktionen mit dem Dateinamen durchzuführen, den Benutzer aber nicht zu warnen.

OFN_SHAREWARN	Dies ist die Standardantwort, die als 0 definiert ist. Weist das Dialogfenster an, eine Standard-Warnmeldung auszugeben.

Siehe auch: *Struktur TData, ShareViMsgId*

Antworttabelleneinträge

Die *TOpenSaveDialog*-Antworttabelle hat keine Einträge.

Struktur TOpenSaveDialog::TData — opensave.h

Die *TOpenSaveDialog*-Struktur enthält Informationen über die Benutzerauswahl zum Öffnen oder Speichern von Dateien. Genauer gesagt speichert diese Struktur einen benutzerdefinierten Dateinamensfilter, eine Dateierweiterung, den Dateinamen, das Stammverzeichnis für die Anzeige von Dateinamen, jegliche Fehlercodes sowie Dateiattribute, die, beispielsweise, festlegen, ob es sich um eine schreibgeschützte Datei handelt. Die Klassen *TFileOpenDialog* und *TFileSaveDialog* verwenden die in dieser Struktur gespeicherten Informatioen, wenn eine Datei geöffnet oder gespeichert wird.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TData(DWORD flags=0, char* filter=0, char* customFilter=0,
          char* initialDir=0, char* defExt=0);
```

Konstruiert eine TOpenSaveDialog::TData-Struktur.

Destruktor

```
~TData();
```

Entfernt eine TOpenSaveDialog::TData-Struktur.

Siehe auch: ::CommDlgExtendedError, ::OpenFileName, TEditFile::FileData

Öffentliche Datenelemente

CustomFilter

```
char* CustomFilter;
```

CustomFilter speichert das vom Benutzer festgelegte Dateifilter, wie zum Beispiel *.CPP.

DefExt

```
char* DefExt
```

DefExt speichert die Standarderweiterung des Dateinamens.

Error

```
uint32 Error;
```

Error enthält einen oder mehrere der folgenden *CommDlgExtendedError*-Codes:

Konstante	Bedeutung
CDERR_DIALOGFAILURE	Dialogfenster konnte nicht erzeugt werden.
CDERR_LOCKRESOURCEFAILURE	Eine angegebene Ressource konnte nicht gesperrt werden.
CDERR_LOADRESFAILURE	Eine angegebene Ressource konnte nicht geladen werden.
CDERR_LOADSTRFAILURE	Ein angegebener String konnte nicht geladen werden.

FileName

```
char* FileName;
```

Enthält den Namen der Datei, die geöffnet oder gespeichert werden soll.

Filter

```
char* Filter;
```

Filter enthält den für die Anzeige von Dateinamen zunächst zu benutzenden Dateifilter.

FilterIndex

```
int FilterIndex;
```

FilterIndex zeigt an, welcher Filter für die Anzeige von Dateinamen zunächst zu benutzen ist.

Flags

```
uint32 Flags;
```

Flags enthält eine oder mehrere der folgenden Windows-API-Konstanten:

Konstante	Bedeutung
OFN_HIDEREADONLY	Verbirgt das nur zum Lesen dienende Markierungsfeld.
OFN_FILEMUSTEXIST	Gestattet dem Benutzer nur die Eingabe bestehender Dateinamen im Eingabefeld. Wenn ein unültiger Dateiname eingegeben wird, löst das die Anzeige einer Warnmeldung aus.
OFN_PATHMUSTEXIST	Gestattet dem Benutzer nur die Eingabe gültiger Pfadnamen. Bei der Eingabe eines ungültigen Pfadnamens wird eine Warnmeldung angezeigt.
OFN_NOVALIDATE	Führt keine Prüfung des Dateinamens durch und benötigt für die Gültigkeitsprüfung den Besitzer einer abgeleiteten Klasse.
OFN_NOCHANGEDIR	Setzt das Verzeichnis vom Dialogbeginn wieder als aktuelles Verzeichnis.
OFN_ALLOWMULITSELECT	Erlaubt Mehrfachauswahl in der Dateiliste.
OFN_CREATEPROMPT	Fragt an, ob der Benutzer eine neue Datei erzeugen möchte.
OFN_EXTENSIONDIFFERENT	Zeigt an, daß der Benutzer einen Dateinamen eingegeben hat, der von der Festlegung in *DefExt* abweicht. Diese Botschaft wird an die aufrufende Funktion zurückgegeben.
OFN_NOREADONLYRETURN	Die zurückgelieferte Datei steht nicht in einem schreibgeschützten Verzeichnis und das Attribut für reinen Lesezugriff ist nicht gesetzt. Diese Botschaft wird an die aufrufende Funktion zurückgegeben.
OFN_NOTESTFILECREATE	Die Datei wird nach dem Schließen des Dialogfensters erzeugt. Wenn die Anwendung dieses Flag setzt, wird keine Überprüfung auf Schreibschutz, volle Festplatte, unverriegeltes Laufwerk oder Netzwerkzugriff durchgeführt. Für bestimmte Netzwerkumgebungen sollte dieses Flag gesetzt sein.

OFN_OVERWRITEPROMPT	Das Dialogfenster »Speichern unter« fragt den Benutzer, ob eine existierende Datei überschrieben werden kann.
OFN_SHAREAWARE	Wenn dieses Flag gesetzt ist und der Aufruf zum Öffnen einer Datei wegen einer Schutzverletzung mißlingt, wird der Fehler ignoriert, und das Dialogfenster liefert den angegebenen Dateinamen zurück. Ist das Flag nicht gesetzt, wird die virtuelle Funktion *ShareViolation* aufgerufen. Sie gibt standardmäßig OFN_SHAREWARN zurück oder einen der folgenden Werte: OFN_SHAREFALLTHROUGH — Der Dateiname wird vom Dialogfenster zurückgegeben. OFN_SHARENOWARN — Es werden keine weiteren Aktionen durchgeführt. OFN_SHAREWARN — Der Benutzer empfängt die Standardwarnmeldung für diesen Fehlertyp.
OFN_SHOWHELP	Zeigt die Hilfe-Schaltfläche im Dialogfenster.

InitialDir

```
char* InitialDir;
```

InitialDir enthält das für die Anzeige von Dateinamen zunächst zu verwendende Verzeichnis.

Öffentliche Elementfunktionen

SetFilter

```
void SetFilter(const char*filter = 0);
```

Kopiert die Liste der Filter für die Anzeige von Dateinamen.

Klasse TOutStream — docview.h

TOutStream ist eine von *TStream* und *ostream* abgeleitete Basisklasse, die zur Erzeugung von Ausgabe-Speicherstreams für ein Dokument verwendet wird.

Öffentlicher Konstruktor

Konstruktor

```
TOutStream(TDocument& doc, const char far* name, int mode);
```

Konstruiert ein *TOutStream*-Objekt. *doc* bezeichnet das Dokumentobjekt, *name* ist der benutzerdefinierte Name des Streams, *mode* sein Öffnungsmodus.

Siehe auch: *TInStream, ofXXXX document open* enum, *shdocument sharing* enum

Klasse TPaint DC — dc.h

Eine Gerätekontextklasse, die aus *TWindowDC* abgeleitet ist, und Rahmenfunktionen für Zeichnen-Anfang- und Zeichnen-Ende-Aufrufe zur

Verwendung in einer WM_PAINT-Kommandofunktion zur Verfügung stellt.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TPaintDC(HWND wnd);
```

Erzeugt ein *TPaintDC*-Objekt mit dem angegebenen (owned) Fenster. Das Datenelement *Wnd* ist auf *wnd* gesetzt.

Siehe auch: *TWindowDC::Wnd, TDC::TDC*

Destruktor

```
~TPaintDC();
```

Entfernt dieses Objekt.

Öffentliche Datenelemente

Ps

```
PAINTSTRUCT Ps;
```

Die Paint-Struktur, die an dieses *TPaintDC*-Objekt gebunden ist.

Siehe auch: *PAINTSTRUCT*

Klasse TPalette — gdiobjec.h

TPalette ist die aus *TGdiObject* abgeleitete GDI-Palettenklasse (GDI = Graphics Device Interface, grafische Geräteschnittstelle). Die Konstruktoren von *TPalette* erzeugen Paletten, entweder aufgrund dafür vorgesehener Informationen oder indirekt durch Verwendung der verschiedenen Farbtabellen, die von DIB-Konstanten benutzt werden.

Öffentliche Konstruktoren

Konstruktor

```
TPalette(HPALETTE handle, TAutoDelete autoDelete = NoAutoDelete);
```

Erzeugt ein *TPalette*-Objekt und setzt das Datenelement *Handle* auf das gegebene, eingelesene *handle*. Das Datenelement *ShouldDelete* ist auf **false** voreingestellt; so ist sichergestellt, daß das eingelesene Handle nicht gelöscht wird, wenn das C++ Objekt entfernt wird.

Konstruktor

```
TPalette(const TClipboard&);
```

Erzeugt ein *TPalette*-Objekt mit den Werten aus der angegebenen Zwischenablage.

Konstruktor

```
TPalette(const TPalette& palette);
```

Dieser öffentliche Kopierkonstruktor erzeugt eine vollständige Kopie des angegebenen *palette*-Objekts. Beispiel:

```
TPalette myPalette = yourPalette;
```

Konstruktor

```
TPalette(const LOGPALETTE far* logPalette);
```

Erzeugt ein *TPalette*-Objekt aus dem angegebenen *logPalette*-Array

Konstruktor

```
TPalette(const PALETTEENTRY far* entries, int count);
```

Erzeugt ein *TPalette*-Objekt mit der in *count* festgesetzten Anzahl von Einträgen aus dem angegebenen Array *entries*.

Konstruktor

```
TPalette(const BITMAPINFO far* info, uint flags = 0);
```

Erzeugt ein *TPalette*-Objekt aus der Farbtabelle entsprechend der angegebenen BITMAPINFO-Struktur. Dieser Konstruktor kann nur für Bitmaps mit 2, 16 oder 256 Farben verwendet werden. Für andere Bitmaps, einschließlich 24-Bit DIBs, wird ein 0-Handle zurückgegeben.

Konstruktor

```
TPalette(const BITMAPCOREINFO far* core, uint flags = 0);
```

Nur für Presentation Manager 1.x-DIBs. Erzeugt ein *TPalette*-Objekt aus der Farbtabelle entsprechend der BITMAPCOREINFO-Struktur. Dieser Konstruktor kann nur für Bitmaps mit 2, 16 oder 256 Farben verwendet werden. Für andere Bitmaps, einschließlich 24-Bit DIBs, wird ein 0-Handle zurückgegeben. Es ist zu beachten, daß jede Farbe einer PM 1.x-Tabelle vorhanden sein muß, da es kein *ClrUsed*-Feld im DIB-Vorspann gibt.

Konstruktor

```
TPalette(const TDib& dib, uint flags = 0);
```

Erzeugt ein *TPalette*-Objekt aus dem angegebenen DIB-Objekt. Das *flags*-Argument repräsentiert die Werte der Datenstruktur, die zur Erzeugung der Palette verwendet werden.

Konstruktor

```
TPalette();
```

(Nur Presentation Manager.) Erzeugt ein Objekt *TPalette*, das die Standard-Systempalette verwendet.

Siehe auch: *TClipboard::GetClipboardData, TGdiObject::Handle, TGdiObject::ShouldDelete, TPalette::GetPaletteEntries,* Struktur *BITMAPCOREINFO,* Struktur *BITMAPINFO,* Struktur *LOGPALETTE,* Struktur *PALETTEENTRY*

Öffentliche Elementfunktionen

AnimatePalette

```
void AnimatePalette(Uint start, uint count,
                    const PALETTEENTRY far* entries);
```

Ersetzt Einträge dieser logischen Palette aus dem *entries*-Array von PALETTEENTRY-Strukturen. *start* gibt den ersten Eintrag an, der umgesetzt werden soll, *count* die Anzahl der zu animierenden Einträge. Die neuen Einträge werden von Windows sofort in der Systempalette verzeichnet.

Siehe auch: Struktur PALETTEENTRY

GetNearestPaletteIndex

```
uint GetNearestPaletteIndex(TColor color) const;
```

Gibt den Index des Farbeintrags zurück, der die beste Übereinstimmung in dieser Palette mit dem angegebenen *color* aufweist.

Siehe auch: *TColor*

GetNumEntries

```
uint GetNumEntries() const;
```

Gibt die Anzahl der Einträge in dieser Palette zurück oder 0, wenn der Aufruf erfolglos bleibt.

Siehe auch: *TGdiObject::GetObject*

GetObject

```
bool GetObject(uint16 far& numEntries) const;
```

Sucht die Anzahl der Einträge in dieser logischen Palette und setzt den Wert in das *numEntries*-Argument ein. Soll die ganze LOGPALETTE-Struktur gefunden werden, ist *GetPaletteEntries* zu verwenden. Gibt bei erfolgreichem Aufruf **true** zurück, andernfalls **false**.

Siehe auch: *TGdiObject::GetObject, TPalette:GetPaletteEntries,* struct *LOGPALETTE*

GetPaletteEntries

```
uint GetPaletteEntries(uint16 start, uint16 count, PALETTEENTRY far*
entries)
                                const;
```

Ermittelt einen Bereich von Einträgen in dieser logischen Palette und setzt sie in das *entries*-Array ein. *start* gibt den ersten zu ermittelnden Eintrag an, *count* die Gesamtzahl der Einträge, die ermittelt werden sollen. Gibt die Anzahl der tatsächlich festgestellten Einträge zurück oder 0, wenn der Aufruf erfolglos bleibt.

Siehe auch: Struktur *PALETTEENTRY*

GetPaletteEntry

```
uint GetPaletteEntry(uint16 index, PALETTEENTRY far& entry) const;
```

Ermittelt den Eintrag bei *index* in dieser logischen Palette und setzt ihn in das *entries*-Array ein. Gibt die Anzahl der tatsächlich ermittelten Einträge zurück: 1 bei erfolgreichem Aufruf, 0 bei Mißerfolg.

Siehe auch: *TPalette::SetPaletteEntry,* Struktur *PALETTEENTRY*

operator<<

```
TClipboard& operator<< (TClipboard& clipboard, TPalette& palette);
```

Kopiert die angegebene *palette* in das angegebene *clipboard*-Argument. Gibt eine Referenz auf die resultierende Zwischenablage zurück, was ein normales Verketten (chaining) von **<<** ermöglicht.

Siehe auch: *TClipboard*

operator HPALETTE()

```
operator HPALETTE() const;
```

Typumwandlungssperator. Konvertiert das Handle dieser Palette in den Typ *HPALETTE*, den Windows-Datentyp, der das Handle zu einer logischen Palette darstellt.

ResizePalette

```
bool ResizePalette(uint numEntries);
```

Ändert die Größe dieser logischen Palette auf die in *numEntries* festgelegte Anzahl von Einträgen. Gibt bei erfolgreichem Aufruf **true** zurück, andernfalls **false**.

Siehe auch: *TPalette::AnimatePalette*

SetPaletteEntries

```
uint SetPaletteEntries(uint16 start, uint16 count, const Paletteentry
far* entries);
```

Setzt die RGB-Farbwerte in dieser Palette aus dem *entries*-Array von PALETTENTRY-Strukturen. *start* gibt den ersten Eintrag an, der umgesetzt werden soll, *count* die Anzahl der zu animierenden Einträge. Gibt die Anzahl der tatsächlich gesetzten Einträge zurück, oder 0 bei Mißerfolg.

Siehe auch: *Struktur PALETTEENTRY*

SetPaletteEntry

```
uint SetPaletteEntry(uint16 index, const PALETTEENTRY far& entry);
```

Setzt den RGB-Farbwert bei *index* in dieser Palette aus dem *entry*-Argument. *start* gibt den ersten zu animierenden Eintrag an, *count* die Anzahl der Einträge, die animiert werden sollen. Gibt bei Erfolg 1 zurück (die Anzahl der tatsächlich gesetzten Einträge), 0 bei Mißerfolg.

Siehe auch: *Struktur PALETTEENTRY*

ToClipboard

```
void ToClipboard(TClipboard& clipboard);
```

Verlagert diese Palette auf das Zielargument *clipboard*. Wenn eine Kopie in die Zwischenablage aufgenommen werden soll, muß sie erst mit *TPalette(myPalette).ToClipboard();* angefertigt werden. *ToClipboard* sollte *delete* auf **false** setzen, so daß das Objekt in der Zwischenablage nicht gelöscht wird. Das Handle steht für Prüfzwecke weiter zur Verfügung.

Siehe auch: *TClipboard::SetClipboardData*

UnrealizeObject

```
bool UnrealizeObject();
```

Weist die grafische Geräteschnittstelle an, beim nächsten Aufruf von *RealizePalette(HDC)* oder *TDC::RealizePalette()* die logische Palette wieder vollständig auf der Systempalette abzubilden. Liefert **true** zurück, wenn der Aufruf erfolgreich war, andernfalls **false**.

Siehe auch: *TDC::RealizePalette*

Geschützte Elementfunktionen

Create

```
void Create(const BITMAPINFO far* info, uint flags);
void Create(const BITMAPCOREINFO far* core, uint flags);
```

Setzt die Werte in dieser Palette aus der gegebenen Bitmap-Struktur. Diese Funktionen werden üblicherweise nicht direkt aufgerufen, sondern durch den Konstruktor.

Siehe auch: Struktur *TBITMAPCOREINFO*, Struktur *BITMAPINFO*

Klasse TPaletteEntry color.h

TPaletteEntry ist eine Unterstützungsklasse, die aus der Struktur *tagPALETTEENTRY* abgeleitet ist. Diese ist folgendermaßen definiert:

```
typedef struct tagPALETTEENTRY {
      uint8  peRed;
      uint8  peGreen;
      uint8  peBlue;
      uint8  peFlags;
  } PALETTEENTRY;
```

peRed, *peGreen* und *peBlue* legen die Intensitätswerte des jeweiligen Paletteneintrags für Rot, Grün und Blau fest.

Das *peFlags*-Element kann die folgenden Werte annehmen:

Wert	Bedeutung
PC_EXPLICIT	Legt fest, daß das niedere Datenwort des Eintrags in der logischen Palette einen Index der Hardware-Palette bezeichnet. Dieses Flag ermöglicht es der Anwendung, den Paletteninhalt des Anzeigegerätes darzustellen.
PC_NOCOLLAPSE	Legt fest, daß die Farbe in einem nicht benutzten Eintrag der Systempalette abgelegt wird, statt an eine existierende Farbe der Systempalette angepaßt zu werden. Wenn es keine freien Einträge in der Systempalette gibt, wird die Farbe normal angepaßt. Sobald die Farbe in die Systempalette aufgenommen ist, können Farben in anderen logischen Paletten daran angepaßt werden.
PC_RESERVED	Legt fest, daß der logische Paletteneintrag für die Animation verwendet wird. So wird verhindert, daß andere Fenster Farben an diesen Eintrag anpassen, da die Farbe häufig wechselt. Ist ein Eintrag in der Systempalette frei, wird die Farbe dort abgelegt. Andernfalls steht sie für die Animation zur Verfügung.

TPaletteEntry wird zusammen mit den Klassen *TPalette* und *TColor* verwendet, um Operationen mit logischen Farbpaletten zu vereinfachen. Es stehen Konstruktoren zur Erzeugung von *TPaletteEntry*-Objekten aus dafür vorgesehenen COLORREF- und RGB-Werten sowie aus *TColor*-Objekten zur Verfügung.

Öffentliche Konstruktoren

Konstruktor

```
TPaletteEntry(int r, int g, int b, int f = 0);
```

Erzeugt ein Paletteneintragsobjekt. Die Parameter *peRed, peGreen, peBlue* und *peFlags* sind auf die jeweiligen Werte von *r, g, b* und *f* gesetzt.

Konstruktor

```
TPaletteEntry(TColor c);
```

Erzeugt ein Paletteneintragsobjekt. Die Werte für *peRed, peGreen, peBlue* und *peFlags* sind jeweils auf *c.Red(), c.Green(), c. Blue()* und *c.Flags* gesetzt.

Siehe auch: Struktur *tagPALETTEENTRY, TPaletteEntry(TColorc), TColor::Red, TColor::Green, TColor::Blue*

Klasse TPen gdiobjec.h

TPen ist aus *TGdiObject* abgeleitet. Das Stiftwerkzeug der grafischen Geräteschnittstelle ist darin eingekapselt. Stifte können aus spezifischen Informationen oder indirekt konstruiert werden. *TPen* baut auf dem Basisdestruktor *~TGdiObject* auf.

Öffentliche Datenelemente

enum TStockId

```
enum TStockId(Null, Black, White);
```

Zählt die in Windows gespeicherten Stifte auf.

Siehe auch: *TGdiObject::GetStock*

Öffentliche Konstruktoren

Konstruktor

```
TPen(HPEN handle, TAutoDelete autoDelete = NoAutoDelete);
```

Erzeugt ein *TPen*-Objekt und setzt das Datenelement *Handle* auf das angegebene eingelesene *handle*. Das Datenelement *ShouldDelete* ist auf **false** voreingestellt; so ist sichergestellt, daß das eingelesene Handle nicht gelöscht wird, wenn das C++ Objekt entfernt wird.

Siehe auch: *TGdiObject::Handle, TGdiObject::ShouldDelete, ::CreatePenIndirect*

Konstruktor

```
TPen(TColor color, int width=1, int style=PS_SOLID);
```

Erzeugt ein *TPen*-Objekt mit den angegebenen Werten. Das *width*-Argument ist in Geräteeinheiten anzugeben, wenn aber 0 gesetzt ist, wird eine Breite von 1 Pixel angenommen. Setzt *Handle* über einen Aufruf von *CreatePen(style,width,color)* der Windows-API mit den angegebenen Standardwerten. Wenn *color* schwarz oder weiß ist, *width* gleich 1 und *style* gefüllt, wird das Handle eines gespeicherten Stifts zurückgegeben. Die Werte für *style* sind in der folgenden Tabelle aufgeführt.

Wert	Bedeutung
PS_SOLID	Erzeugt einen gefüllten Stift.
PS_DASHED	Erzeugt einen Stift für gestrichelte Linie. Nur gültig , wenn die Stiftbreite in Geräteeinheiten 1 oder weniger beträgt.
PS_DOT	Erzeugt eine gepunktete Linie. Nur gültig, wenn die Stiftbreite in Geräteeinheiten 1 oder weniger ist.
PS_DASHDOT	Dieser Stift zeichnet eine Linie mit abwechselnden Strichen und Punkten. Nur gültig bei einer Stiftbreite von 1 oder weniger in Geräteeinheiten.
PS_DASHDOTDOT	Dieser Stift zeichnet eine Linie mit abwechselnden Strichen und doppelten Punkten. Nur gültig bei einer Stiftbreite von 1 oder weniger in Geräteeinheiten.
PS_NULL	Erzeugt einen Null-Stift.
PS_INSIDEFRAME	Erzeugt einen gefüllten Stift. Wenn dieser Stift in GDI-

Zeichenfunktionen benutzt wird, denen ein Begrenzungsrechteck zugewiesen werden kann, werden die Abmessungen der gezeichneten Figur so verkleinert, daß sie vollständig in das Begrenzungsrechteck paßt; dabei wird die Stiftbreite berücksichtigt.

Siehe auch: *::Create Pen, TColor*

Konstruktor

```
TPen(const LOGPEN far* logPen);
```

Erzeugt ein *TPen*-Objekt aus den angegebenen *logPen*-Werten. Setzt *Handle* über einen Aufruf von *CreatePenIndirect(logPen)* der Windows-API.

Siehe auch: *::CreatePenIndirect*

Konstruktor

```
TPen(DWORD penStyle, DWORD width, const TBrush& brush, DWORD styleCount,
     LPDWORD style);
```

Erzeugt ein *TPen*-Objekt mit den gegebenen Werten. Setzt *Handle* über einen Aufruf von *ExtCreatePen(penStyle, width, &logBrush, styleCount, style)* der Windows-API (*logBrush* gibt den Wert des angegebenen *brush*-Objektes wieder).

Siehe auch: *::ExtCreatePen*

Konstruktor

```
TPen(DWORD penStyle, DWORD width, const LOGBRUSH& logBrush,
     DWORD styleCount, LPDWORD style);
```

Erzeugt ein *TPen*-Objekt mit den angegebenen Werten. *Handle* wird über einen Aufruf der Windows-API-Funktion *ExtCreatePen(penStyle, width, &logBrush, styleCount, style)* gesetzt.

Öffentliche Elementfunktionen

GetObject

```
inline bool GetObject (LOGPEN far& logPen) const;
```

Ermittelt Informationen über dieses Stiftobjekt und setzt sie in die angegebene *LOGPEN*-Struktur ein. Gibt bei erfolgreichem Aufruf **true** zurück, andernfalls **false**.

Siehe auch: *TGdiObject::GetObject*, struct *LOGPEN*

operator HPEN()

```
operator HPEN() const;
```

Typumwandlungsoperator. Konvertiert das *Handle* dieses Stifts in den Typ HPEN (den Windows-Datentyp, der Handles für logische Stifte repräsentiert).

Aufzählungstyp TPicResult — validate.h

TPicResult ist der von der Elementfunktion *Picture* von *TPXPictureValidator* zurückgelieferte Ergebnistyp. Er enthält einen der folgenden Typen: prIncomplete, prEmpty, prError, prSyntax, prAmbiguous, prIncompNoFill.

enum TPicResult;

Siehe auch: *TPXPictureValidator*

Aufzählungstyp TPlacement — gadgetwi.h

Zählt die Positionierung eines Gadget auf: entweder vor oder hinter einem anderen Gadget.

```
enum TPlacement;
```

Siehe auch: *TGadgetWindow::PositionGadgets*

Klasse TPopupMenu — menu.h

TPopupMenu erzeugt ein leeres Popup-Menü, das einem vorhandenen Fenster oder Popup-Menü hinzugefügt werden kann. Zusätzliche Informationen zu Menü-Objekten finden sich in Kapitel 9 des ObjectWindows *Programmierhandbuchs*.

Öffentliche Konstruktoren

Konstruktor

```
TPopupMenu(TAutoDelete autoDelete = AutoDelete);
```

Erzeugt ein leeres Popup-Menü.

Konstruktor

```
TPopupMenu(HMENU handle, TAutoDelete autoDelete = NoAutoDelete);
```

Alias-Konstruktor für ein Popup-Menü.

Öffentliche Elementfunktionen

TrackPopupMenu

```
bool TrackPopupMenu(uint flags, int x, int y, int rsvd, HWND wnd,
                    TRect* rect = 0);
```

Erlaubt der Anwendung, ein Popup-Menü an der angegebenen Stelle im angegebenen Fenster zu erstellen. *flags* gibt eine Bildschirmposition an und kann einer der Windows-API *TPM_xxxx*-Werte (TPM_CENTERALIGN, TPM_LEFTALIGN, TPM_RIGHTALIGN, TPM_LEFTBUTTON oder TPM_RIGHTBUTTON) sein. *wnd* ist das Handle desjenigen Fensters, das Botschaften über das Menü erhält. *x* gibt die horizontale Position der linken Seite des Menüs in Bildschirmkoordinaten an. *y* gibt die vertikale Position der Oberkante des Menüs in Bildschirmkoordinaten an (z.B. besagt 0,0, daß die linke Ecke eines Menüs sich in der oberen linken Ecke des Bildschirms befindet. *rect* bezeichnet die Fläche, in der der Benutzer klicken kann, ohne das Menü zu verlassen.

Siehe auch: *::TrackPopupMenu*

TrackPopupMenu

```
bool TrackPopupMenu(uint flags, TPoint& point, int rsvd, HWND wnd,
                    TRect* rect = 0);
```

Diese Funktion entspricht dem eben beschriebenen *TrackPopupMenu*. Hier werden die *x*- und *y*-Positionen jedoch in *Punkt* (*point*) angegeben.

Siehe auch: *TPM_xxxx (Windows API)*

Klasse TPreviewPage — preview.h

TPreviewPage zeigt eine Seite eines Dokuments in einem Voranzeigefenster an. *TPreviewPage* greift auf *TPrintPreviewDC* und *TPrintout* zu, um die benötigten Informationen zu erhalten. Insbesondere generiert es ein *TPrintPreviewDC* aus dem in *Paint* zur Verfügung gestellten Fenster-DC und übergibt dieses *TPrintPreviewDC*-Objekt an die *SetPrintParams*-Elementfunktion von *TPrintout*. PPRINT.CPP (Verzeichnis OWLAPI\PRNTPREV) zeigt das folgende Beispiel für ein Druckvoranzeigefenster an:

Öffentliche Konstruktoren

Konstruktor

```
TPreviewPage (TWindow* parent, TPrintout& printout, TPrintDC& prndc,
              TSize& printExtent, int pagenum = 1);
```

Baut ein *TPreviewPage*-Objekt mit *parent* als übergeordnetem Fenster auf. *printout* ist die Referenz zu dem *TPrintPreviewDC*-Objekt, *printExtent* ist das Maß (Breite und Höhe) der gedruckten Seite in logischen Einheiten, und

pagenum bezeichnet die Seitennummer der Voranzeigeseite. Initialisiert die Attribut-Bits in *Attr.Style*. *TPreviewPage* erhält dadurch die Attribute eines sichtbaren untergeordneten Fensters mit einer dünnen Begrenzung. Setzt die Hintergrundfarbe des Voranzeigeseite-Fensters auf weiß.

Öffentliche Elementfunktionen

Paint

```
void Paint(TDC& dc, bool, TRect& clip);
```

Zeigt die Seite im Voranzeigefenster an. *Paint* ruft verschiedenen *TPrintout*-Elementfunktionen auf, um die Attribute der Voranzeigeseite festzulegen. Dann wird das Druckobjekt für die Voranzeige angepaßt. *Paint* bestimmt, ob die Seite in das Voranzeigefenster paßt oder ob Clipping (Beschneiden) nötig ist. Schließlich gibt *Paint* Informationen zum Clipping und zur abschnittweisen Bearbeitung an die *PrintPage*-Funktion *TPrintout* weiter, die zur Darstellung der Seite im Voranzeigefenster aufgerufen wird.

Siehe auch: *TPrintout::BeginPrinting, TPrintout::EndPrinting, TPrintout::PrintPage*

SetPageNumber

```
void SetPageNumber(int newNum);
```

Setzt *newNum* auf die Seitennummer der aktuellen Seite, die im Voranzeigefenster dargestellt wird.

Geschützte Datenelemente

PageNum

```
int PageNum;
```

Nummer der Seite, die im Voranzeigefenster dargestellt wird.

PrintDC

```
TPrintDC& PrintDC;
```

PrintDC& ist eine ID für den Gerätekontext, die für den Druck verwendet werden muß.

PrintExtent

```
TSize PrintExtent;
```

Enthält den Umfang (Breite und Höhe) der Seite in logischen Einheiten.

Printout

```
TPrintout& Printout;
```

Enthält eine Referenz auf das *TPrintout*-Objekt.

Geschützte Elementfunktionen

EvSize

```
void EvSize(uint sizeType, TSize& size);
```

Macht das gesamte Fenster ungültig, wenn die Größe der im Voranzeigefenster dargestellten Seite sich ändert.

Antworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_WM_SIZE	EvSize

Klasse TPrintDC dc.h

TPrintDC ist von *TDC* abgeleitet und erlaubt den Zugriff auf einen Drucker.

Öffentliche Konstruktoren

Konstruktor

```
TPrintDC(HDC handle, TAutoDelete autoDelete = NoAutoDelete);
```

Erzeugt ein *TPrint*-Objekt für das durch DC bezeichnete *Handle*.

Siehe auch: *TAutoDelete, TDC::TDC*

Konstruktor

```
TPrintDC(const char far* driver, const char far* device, const char far*
        output, const DEVMODE far* initData);
```

Erzeugt ein *TPrint*-Objekt unter Verwendung des Druckertreibers, des Geräts, der Ausgabe und der Daten aus der DEVMODE-Struktur.

Öffentliche Elementfunktionen

AbortDoc

```
int AbortDoc();
```

Bricht den aktuellen Druckauftrag auf diesem Drucker ab und löscht alles, was seit dem letzten Aufruf von *StartDoc* gezeichnet wurde. *AbortDoc* ruft den benutzerdefinierten Funktionssatz mit *TPrintDC::SetAbortProc()* auf , um einen Druckauftrag wegen eines Fehlers oder durch Benutzereingriff abzubrechen. *TPrintDC::EndDoc* sollte benutzt werden, um einen erfolgreich abgeschlossenen Druckauftrag zu beenden.

Bei Erfolg gibt *AbortDoc* einen positiven Wert oder Null zurück; andernfalls wird ein negativer Wert zurückgegeben.

Bitte beachten Sie, daß Dokumente in früheren als Win 3.1-Anwendungen durch Aufruf von *Escape()* mit dem Escape-Wert ABORTDOC abgebrochen werden. Ab Version 3.1 wird dieses Escape-Verfahren durch die *::AbortDoc*-Funktion abgelöst. *TPrintCD::AbortDoc* wählt automatisch den entsprechenden Aufruf.

Siehe auch: *::AbortDoc, TPrintDC::EndDoc, TPrintDC::SetAbortProc, TPrintDC::Escape, ABORTDOC*

BandInfo

```
int BandInfo(TBandInfo& bandInfo);
```

Liefert Informationen zu den Möglichkeiten der abschnittsweisen Bearbeitung durch dieses Gerät und kopiert sie in die vorgegebene *bandInfo*-Struktur. Gibt 1 zurück, wenn der Aufruf erfolgreich ist; gibt 0 zurück, wenn der Aufruf nicht erfolgreich war oder wenn abschnittsweise Bearbeitung von diesem Gerät nicht unterstützt wird.

Siehe auch: *struct TBandInfo, TPrintDC::Escape, BANDINFO*

DeviceCapabilities

```
static uint32 DeviceCapabilities(const char far* driver, const char far*
                                 device, const char far* port,
                                 int capability, char far* output=0,
                                 LPDEVMODE devmode=0);
```

Liefert Daten zur angegebenen *capability* des ausgewählten Druckertreibers (*driver*), des Geräts (*device*) und der Schnittstelle (*port*) und schreibt die Ergebnisse in das *output*-**char**-Array. Die Treiber-, Geräte- und Schnittstellennamen müssen nullterminierte Strings sein. Das *devmode*-Argument zeigt auf ein DEVMODE-**struct**. Wenn *devmode* 0 (die Voreinstellung) ist, liefert *DeviceCapabilities* die aktuellen voreingestellten Initialisierungswerte für den angegebenen Druckertreiber. Andernfalls liefert es die in der DEVMODE-Struktur enthaltenen Werte. Das Format des *output*-Array hängt von den abgefragten Fähigkeiten ab. Wenn *output* 0 (die Voreinstellung) ist, gibt *DeviceCapabilities* die Anzahl der im *output*-Array benötigten Bytes zurück. Mögliche Werte für *capability* sind:

Wert	Beschreibung
DC_BINNAMES	Die Funktion zählt die Papierschächte des angegebenen Geräts. Wenn diese Konstante durch einen Gerätetreiber unterstützt wird, ist das *output*-Array eine Datenstruktur, die zwei Elemente enthält. Das erste Element ist ein Array, das gültige Papierschächte angibt:

```
short BinList[cBinMax]
```

Das zweite Element ist ein Zeichenstring-Array, das die Schachtnamen angibt:

```
char PaperNames[CBinMax][cchBinName]
```

Das *output*-Array ist leer und der zurückgegebene Wert ist NULL, wenn ein Gerätetreiber diesen Wert nicht unterstützt.

Wenn *output* NULL ist, gibt der zurückgelieferte Wert die Anzahl der unterstützten Schächte an.

```
DC_BINS
```

Die Funktion liefert eine Konstantenliste, die die verfügbaren Schächte angibt und kopiert die Liste in das *output*-Array. Wenn dieses Array NULL

ist, gibt die Funktion die Anzahl der unterstützten Schächte zurück. Folgende Schachtbezeichner können zurückgegeben werden:

```
DMBIN_AUTO
DMBIN_CASSETTE
DMBIN_ENVELOPE
DMBIN_ENVMANUAL
DMBIN_FIRST
DMBIN_LARGECAPACITY
DMBIN_LARGEFMT
DMBIN_LAST
DMBIN_LOWER
DMBIN_MANUAL
DMBIN_MIDDLE
DMBIN_ONLYONE
DMBIN_SMALLFMT
DMBIN_TRACTOR
DMBIN_UPPER
```

```
DC_DRIVER
```

Die Funktion gibt die Versionsnummer des Treibers zurück.

```
DC_DUPLEX
```

Die Funktion gibt den Grad der Unterstützung für doppelseitigen Ausdruck zurück. Der zurückgegebene Wert ist 1, wenn die Funktion doppelseitigen Ausdruck unterstützt. Wenn nicht, ist der Wert 0.

```
DC_ENUMRESOLUTIONS
```

Die Funktion kopiert eine Liste der verfügbaren Druckerauflösungen in das *output*-Array. Die Auflösungen werden als LONG-Integerwert-Paare kopiert. Der erste Wert des Paares gibt die horizontale Auflösung an, der zweite Wert die vertikale Auflösung. Wenn *output* 0 ist, gibt die Funktion die Anzahl der unterstützten Auflösungen zurück.

```
DC_EXTRA
```

Die Funktion gibt die für die geräteabhängigen Daten benötigte Bytezahl zurück. Sie wird an die DEVMODE-Struktur angehängt.

```
DC_FIELDS
```

Die Funktion liefert einen Wert zurück, der angibt, welche Elemente der DEVMODE-Struktur durch den Gerätetreiber gesetzt werden. Dieser Wert kann eine oder mehrere der folgenden Konstanten sein:

```
DM_ORIENTATION
DM_PAPERSIZE
DM_PAPERLENGTH
DM_PAPERWIDTH
DM_SCALE
DM_COPIES
```

```
DM_DEFAULTSOURCE
DM_PRINTQUALITY
DM_COLOR
DM_DUPLEX
DM_YRESOLUTION
DM_TTOPTION
```

`DC_FILEDEPENDENCIES`

Die Funktion liefert eine Dateiliste zurück, die geladen werden muß, wenn der Gerätetreiber installiert wird. Wenn *output* 0 ist und dieser Wert angegeben wird, gibt die Funktion die Anzahl der zu ladenden Dateinamen zurück. Wenn *output* nicht 0 ist, gibt die Funktion die angegebene Anzahl von Dateinamen mit 64 Zeichen zurück.

`DC_MAXEXTENT`

Die Funktion liefert die maximale Papiergröße zurück, die unterstützt wird. Die Abmessungen werden als POINT-Struktur zurückgegeben. Das *x*-Element gibt die maximale Papierbreite an und das *y*-Element die maximale Papierlänge.

`DC_MINEXTENT`

Die Funktion liefert die minimale Papiergröße zurück, die unterstützt wird. Die Abmessungen werden als POINT-Struktur zurückgegeben. Das *x*-Element gibt die minimale Papierbreite an und das *y*-Element die minimale Papierlänge.

`DC_PAPERS`

Die Funktion liefert eine Liste der unterstützten Papierformate und kopiert sie in das *output*-Array. Die Funktion liefert die Anzahl der im Array bezeichneten Größen zurück. Wenn *output* 0 ist, gibt die Funktion die Anzahl der unterstützten Papierformate zurück.

`DC_PAPERSIZE`

Die Funktion liefert die unterstützten Papierformate (in .1-Millimeter-Einheiten) und kopiert sie in das *output*-Array.

`DC_SIZE`

Die Funktion liefert die Größe der DEVMODE-Struktur, die von dem jeweiligen Gerätetreiber benötigt wird.

`DC_VERSION`

Die Funktion gibt die Versionsnummer des Gerätetreibers zurück.

Wenn *DeviceCapabilities* erfolgreich ist, hängt der Rückgabewert von *capability* ab (wie bereits beschrieben wurde). Ansonsten ist der Rückgabewert GDI_ERROR.

Siehe auch: *::DeviceCapabilitiesEx, struct DEVMODE, TDC::GetDeviceCaps*

EndDoc

```
inline int EndDoc();
```

Beendet den aktuellen Druckauftrag auf diesem Drucker. *EndDoc* sollte sofort nach einem erfolgreich abgeschlossenen Druckauftrag aufgerufen werden. *TPrintDC::AbortDoc* sollte zum Beenden eines Druckauftrags durch Fehler oder Benutzereingriff verwendet werden.

Bei Erfolg gibt *EndDoc* einen positiven Wert oder Null zurück. Andernfalls wird ein negativer Wert zurückgegeben.

Anwendungen unter Windows in einer früheren als Version Win 3.1 werden durch Aufruf von *Escape()* mit dem Escape-Wert ENDDOC beendet. Ab Version Win 3.1 wird dieses Escape-Verfahren durch die *::EndDoc*-Funktion abgelöst. *TPrintDC::EndDoc* wählt automatisch den richtigen Aufruf.

Siehe auch: *::EndDoc, TPrintDC::StartDoc, TPrintDC::AbortDC, TPrintDC::Escape, ABORTDOC*

EndPage

```
int EndPage();
```

Teilt dem Gerätetreiber des jeweiligen Druckers mit, daß die Anwendung nicht mehr auf eine Seite schreibt. Bei Erfolg gibt *EndPage* einen positiven Wert oder Null zurück. Andernfalls wird ein negativer Wert zurückgegeben. Mögliche Fehlerwerte sind:

Wert	Beschreibung
SP_ERROR	Allgemeiner Fehler.
SP_APPABORT	Auftrag beendet, da die Druck-Abbruch-Funktion der Anwendung 0 zurückgegeben hat.
SP_USERABORT	Benutzer hat den Auftrag über den Windows Druck-Manager beendet (PRINTMAN.EXE).
SP_OUTOFDISK	Plattenplatz nicht ausreichend für Spooling.
SP_OUTOFMEMORY	Speicher nicht ausreichend für Spooling.

Anwendungen unter Windows in einer früheren Version als Win 3.1 werden durch Aufruf von *Escape()* mit dem Escape-Wert NEWFRAME beendet. Ab Version Win 3.1 wird dieses Escape-Verfahren durch die *::EndPage*-Funktion abgelöst. *TPrintDC::EndPage* wählt automatisch den richtigen Aufruf.

Siehe auch: *::EndPage, TPrintDC::StartPage, TPrintDC::Escape, NEWFRAME*

Escape

```
int Escape(int escape, int count=0, const void* inData=0, void*
                    outData=0);
```

Erlaubt der Anwendung, auf die Fähigkeiten eines bestimmten Gerätes zuzugreifen, die durch das GDI dieses DCs nicht direkt verfügbar sind.

Der *Escape*-Aufruf wird durch das Setzen eines Mnemo-Wertes im *escape*-Argument bestimmt. Unter Win32 wurde die Verwendung von *Escape* mit entsprechenden *escape*-Werten durch spezifische Funktionen ersetzt. Die Namen dieser neuen Funktionen basieren auf den entsprechenden *escape*-Mnemos, wie die folgende Tabelle zeigt:

Wert	Maßnahme
ABORTDOC	In Win32 durch *TPrintDC:AbortDoc()* ersetzt.
BANDINFO	In Win32 überholt. Da alle Druckertreiber für Windows Version 3.1 oder später das Text-Flag in jedem Abschnitt setzen, ist dieses Escape nur bei älteren Druckertreibern sinnvoll.
BEGIN_PATH	Keine Änderungen unter Win32. Dieses Escape bezieht sich auf PostScript-Drucker.
CLIP_TO_PATH	Keine Änderungen unter Win32. Dieses Escape bezieht sich auf PostScript-Drucker.
DEVICEDATA	In Win32 ersetzt. Anwendungen sollten das PASSTROUGH-Escape benutzen, das zum gleichen Ergebnis führt.
DRAFTMODE	In Win32 ersetzt. Anwendungen erzielen das gleiche Ergebnis durch Setzen des dmPrintQuality-Elements der DEVMODE-Struktur auf DMRES_DRAFT und Übergabe dieser Struktur an die CreateDC-Funktion.
DRAWPATTERNRECT	Keine Änderungen unter Win32.
ENABLEDUPLEX	In Win32 ersetzt. Anwendungen erzielen das gleiche Ergebnis durch Setzen des dmDuplex-Elements der DEVMODE-Struktur und Übergabe dieser Struktur an die CreateDC-Funktion.
ENABLEPAIRKERNING	Keine Änderungen unter Win32.
ENABLERELATIVEWIDTHS	Keine Änderungen unter Win32.
ENDDOC	In Win32 durch *TPrintDC::EndDoc()* ersetzt.
END_PATH	Keine Änderungen unter Win32. Dieses Escape bezieht sich auf PostScript-Drucker.
ENUMPAPERBINS	In Win32 überholt. Anwendungen können *TPrintDC::DeviceCapabilities()* benutzen, um das gleiche Ergebnis zu erzielen.
ENUMPAPERMETRICS	In Win32 überholt. Anwendungen können *TPrintDC::DeviceCapabilities()* benutzen, um das gleiche Ergebnis zu erzielen.
EPSPRINTING	Keine Änderungen unter Win32. Dieses Escape bezieht sich auf PostScript-Drucker.
EXT_DEVICE_CAPS	In Win32 überholt. Anwendungen können *TDC::GetDeviceCaps()* benutzen, um das gleiche Ergebnis zu erzielen. Dieses Escape bezieht sich auf PostScript-Drucker.
EXTTEXTOUT	In Win32 überholt. Anwendungen können *TDC::ExTextOut()* benutzen, um das gleiche Ergebnis zu erzielen. Dieses Escape wird vom PCL-Treiber in Version 3.1 nicht unterstützt.
FLUSHOUTPUT	Aus Win32 entfernt.

GETCOLORTABLE Aus Win32 entfernt.

GETEXTENDEDTEXTMETRICS
Keine Änderungen unter Win32. Die Unterstützung für dieses Escape ändert sich möglicherweise in zukünftigen Windows-Versionen.

GETEXTENTTABLE In Win32 überholt. Anwendungen können *::GetCharWidth* benutzen, um das gleiche Ergebnis zu erzielen. Dieses Escape wird von PCL- oder PostScript-Treibern in Version 3.1 nicht unterstützt.

GETFACENAME Keine Änderungen unter Win32. Dieses Escape bezieht sich auf PostScript-Drucker.

GETPAIRKERNTABLE Keine Änderungen unter Win32.

GETPHYSPAGESIZE Keine Änderungen unter Win32. Die Unterstützung für dieses Escape ändert sich möglicherweise in zukünftigen Windows-Versionen.

GETPRINTINGOFFSET Keine Änderungen unter Win32. Die Unterstützung für dieses Escape ändert sich möglicherweise in zukünftigen Windows-Versionen.

GETSCALINGFACTOR Keine Änderungen unter Win32. Die Unterstützung für dieses Escape ändert sich möglicherweise in zukünftigen Windows-Versionen.

GETSETPAPERBINS In Win32 ersetzt. Anwendungen können durch den Aufruf von *TPrintDC::DeviceCapabilities()* zur Feststellung der Anzahl der Papierschächte das gleiche Ergebnis zu erzielen. Durch Aufruf von *::ExtDeviceMode* wird der aktuelle Schacht ausgegeben. Dann wird das *dmDefaultSource*-Element der DEVMODE-Struktur gesetzt und diese Struktur an die *CreateDC*-Funktion übergeben. GETSETPAPERBINS wechselt den Papierschacht nur für den aktuellen Gerätekontext. In einem neuen Gerätekontext wird der Standard-Papierschacht des Systems benutzt, bis der Schacht für diesen Gerätekontext explizit geändert wird.

GETSETPARAMETRICS
In Win32 überholt. Anwendungen können *TPrintDC::DeviceCapabilities()* und *::ExtDeviceMode()* benutzen, um das gleiche Ergebnis zu erzielen.

GETSETPAPERORIENT In Win32 überholt. Anwendungen erzielen das gleiche Ergebnis durch Setzen des *dmOrientation*-Elements der DEVMODE-Struktur und Übergabe dieser Struktur an die *CreateDC*-Funktion. Dieses Escape wird vom PCL-Treiber in Version 3.1 nicht unterstützt.

GETSETSCREENPARAMS
Keine Änderungen unter Win32.

GETTECHNOLOGY Keine Änderungen unter Win32. Die Unterstützung für dieses Escape ändert sich möglicherweise in zukünftigen Windows-Versionen. Dieses Escape wird vom PCL-Treiber in Version 3.1 nicht unterstützt.

GETTRACKKERNTABLE
Keine Änderungen unter Win32.

GETVECTORBRUSHSIZE
Keine Änderungen unter Win32. Die Unterstützung für dieses Escape ändert sich möglicherweise in zukünftigen Windows-Versionen.

GETVECTORPENSIZE Keine Änderungen unter Win32. Die Unterstützung für dieses Escape ändert sich möglicherweise in zukünftigen Windows-Versionen.

MFCOMMENT Keine Änderungen unter Win32.

NEWFRAME Keine Änderungen unter Win32. Anwendungen sollten statt dieses Escapes *::StartPage()* und *::EndPage()* verwenden. Die Unterstützung für dieses Escape ändert sich möglicherweise in zukünftigen Windows-Versionen.

NEXTBAND Keine Änderungen unter Win32. Die Unterstützung für dieses Escape ändert sich möglicherweise in zukünftigen Windows-Versionen.

PASSTROUGH Keine Änderungen unter Win32.

QUERYESCAPESUPPORT
Keine Änderungen unter Win32.

RESTORE_CTM Keine Änderungen unter Win32. Dieses Escape bezieht sich auf PostScript-Drucker.

SAVE_CTM Keine Änderungen unter Win32. Dieses Escape bezieht sich auf PostScript-Drucker.

SELECTPAPERSOURCE
In Win32 überholt. Anwendungen können *TPrintDC::DeviceCapabilities()* benutzen, um das gleiche Ergebnis zu erzielen.

SETABORTPROC In Win32 ersetzt durch *::SetAbortProc()*. Vgl. *TPrintDC::SetAbortProc*.

SETALLJUSTVALUES Keine Änderungen unter Win32. Die Unterstützung für dieses Escape ändert sich möglicherweise in zukünftigen Windows-Versionen. Dieses Escape wird vom PCL-Treiber in Version 3.1 nicht unterstützt.

SET_ARC_DIRECTION Keine Änderungen unter Win32. Dieses Escape bezieht sich auf PostScript-Drucker.

SET_BACKGROUND_COLOR
Keine Änderugen unter Win32. Anwendungen sollten statt dieses Escapes *::SetBkColor()* verwenden. Die Unterstützung für dieses Escape ändert sich möglicherweise in zukünftigen Windows-Versionen.

SET_BOUNDS Keine Änderungen unter Win32. Dieses Escape bezieht sich auf PostScript-Drucker.

SETCOLORTABLE Keine Änderugen unter Win32. Die Unterstützung für dieses Escape ändert sich möglicherweise in zukünftigen Windows-Versionen.

SETCOPYCOUNT In Win32 ersetzt. Anwendungen sollten zur Feststellung der maximalen Kopienzahl, die ein Gerät erstellen kann, *TPrintDC::DeviceCapabilities()* unter Angabe von DC_COPIES für den *nIndex*-Parameter aufrufen. Die Anwendung kann dann durch Übergabe eines Zeigers auf die DEVMODE-Struktur and die *CreateDC*-Funktion die Kopienanzahl setzen.

SETKERNTRACK Keine Änderungen unter Win32.

SETLINECAP Keine Änderungen unter Win32. Dieses Escape bezieht sich auf PostScript-Drucker.

SETLINEJOIN Keine Änderungen unter Win32. Dieses Escape bezieht sich auf PostScript-Drucker.

SETMITERLIMIT Keine Änderungen unter Win32. Dieses Escape bezieht sich auf PostScript-Drucker.

SET_POLY_MODE	Keine Änderungen unter Win32. Dieses Escape bezieht sich auf PostScript-Drucker.
SET_SCREEN_ANGLE	Keine Änderungen unter Win32.
STARTDOC	In Win32 ersetzt. Anwendungen sollten statt dieses Escapes *::StartDoc* aufrufen.
TRANSFORM_CTM	Keine Änderungen unter Win32. Dieses Escape bezieht sich auf PostScript-Drucker.

Escape-Aufrufe werden übersetzt und an den Drucker-Gerätetreiber geschickt. Der *inData*-Puffer erlaubt die Versorgung mit allen Daten, die für das Escape erforderlich sind. *count* muß auf die Größe (in Bytes) des *inData*-Puffers gesetzt werden. Wenn keine Eingabedaten erforderlich sind, sollten *inData* und *count* auf den Standard-Wert 0 gesetzt werden. Analog muß ein *outData*-Puffer für die *Escape*-Aufrufe, die Daten liefern, verfügbar gemacht werden. Wenn das Escape keine Ausgaben liefert, setzen Sie *outData* auf den Standard-Wert 0.

Siehe auch: *::Escape*

NextBand

```
int NextBand(TRect& rect);
```

Teilt dem Gerätetreiber des Druckers mit, daß die Anwendung nicht mehr in einen Abschnitt schreibt. Der Gerätetreiber schickt den fertiggestellten Abschnitt an den Print-Manager und kopiert die Koordinaten des nächsten Abschnitts in das Rechteck, das durch *rect* angegeben wird.

Bei Erfolg gibt *NextBand* einen positiven Wert oder Null zurück; andernfalls wird ein negativer Wert zurückgegeben. Mögliche Fehlerwerte sind:

Wert	Beschreibung
SP_ERROR	Allgemeiner Fehler.
SP_APPABORT	Auftrag beendet, da die Druck-Abbruch-Funktion der Anwendung 0 zurückgegeben hat.
SP_USERABORT	Benutzer hat den Auftrag über den Windows Druck-Manager beendet (PRINTMAN.EXE).
SP_OUTOFDISK	Plattenplatz nicht ausreichend für Spooling.
SP_OUTOFMEMORY	Speicher nicht ausreichend für Spooling.

Siehe auch: *TPrintDC::Escape, NEXTBAND, TPrintDC::BandInfo*

QueryAbort

```
bool QueryAbort(int rsvd=0);
```

Versucht, die *AbortProc*-Callback-Funktion des Druckers aufzurufen, um festzustellen, ob ein Druckauftrag abgebrochen werden soll oder nicht. *QueryAbort* gibt entweder den Wert, den *AbortProc* zurückliefert, zurück oder **true**, wenn keine solche Callback-Funktion verfügbar ist. Bei Rückgabe von **true** soll der Druck fortgesetzt werden, und bei Rückgabe von

false soll der Druckauftrag abgebrochen werden. Das *rsvd*-Argument ist ein reservierter Wert, der auf 0 gesetzt werden sollte.

Siehe auch: *::QueryAbort, TPrintDC::SetAbortProc, TPrintDC::AbortDoc, ::AbortProc*

QueryEscSupport

```
uint QueryEscSupport(int escapeNum);
```

Gibt **true** zurück, wenn das durch *escapeNum* bezeichnete Escape auf dem Gerät implementiert wurde. Andernfalls wird **false** zurückgegeben.

Siehe auch: *TPrintDC::Escape, QUERYESCSUPPORT*

SetAbortProc

```
int SetAbortProc(ABORTPROC proc);
```

Richtet das benutzerdefinierte *proc* als Druckabbruch-Funktion für den Drucker ein. Die Funktion wird durch *TPrintDC::AbortDoc* aufgerufen, um einen Druckauftrag während des Spoolings abzubrechen.

Bitte beachten Sie, daß Abbruchfunktionen in früheren als Win 3.1-Anwendungen durch Aufruf von *Escape()* mit dem Escape-Wert SETABORTDOC gesetzt werden. Ab Version 3.1 wird dieses Escape-Verfahren durch die *::SetAbortProc*-Funktion abgelöst. *TPrintDC::SetAbortProc* wählt automatisch den entsprechenden Aufruf.

Bei Erfolg gibt *SetAbortProc* einen positiven Wert (ungleich Null) zurück; andernfalls wird ein negativer Wert (ungleich Null) zurückgegeben.

Siehe auch: *::SetAbortProc, TPrintDC::Escape, SETABORTPROC*

SetCopyCount

```
int SetCopyCount(int requestCount, int& actualCount);
```

Weist *requestCount* die Anzahl der unsortierten Kopien jeder Seite zu, die der Drucker ausgeben soll. Die aktuelle Anzahl der zu druckenden Kopien wird nach *actualCount* kopiert. Die tatsächliche Anzahl wird kleiner als die abgefragte Anzahl sein, wenn letztere das Maximum, das auf diesem Gerät erlaubt ist, übersteigt. Bei Erfolg gibt *SetCopyCount* 1 zurück, ansonsten wird 0 zurückgegeben.

Siehe auch: *TPrintDC::DeviceCapabilities, TPrintDC::Escape, SETCOPYCOUNT*

StartDoc

```
int StartDoc(const char far* docName, const char far* output);
```

Startet einen Druckauftrag für das angegebene Dokument auf dem Drucker-DC. Bei Erfolg gibt *StartDoc* einen positiven Wert zurück: die Auftragskennung des Dokuments. Wenn der Aufruf scheitert, wird der Wert SP_ERROR zurückgegeben. Detaillierte Fehlermeldungen können über *GetLastError* abgerufen werden.

Diese Fuktion ersetzt den früheren *::Escape*-Aufruf durch den Wert STARTDOC.

Siehe auch: *::StartDoc, ::GetJob, ::SetJob, TPrintDC::EndDoc, TPrintDC::Escape*

StartPage

```
int StartPage();
```

Bereitet das Gerät auf den Empfang von Daten vor. Das System deaktiviert *::ResetDC()* zwischen *StartPage*- und *EndPage*-Aufrufen. Anwendungen können deshalb den Gerätemodus - außer an den Seitenbegrenzungen - nicht ändern.

Bei Erfolg gibt *StartPage* einen positiven Wert zurück, ansonsten wird ein negativer Wert oder Null zurückgegeben.

Siehe auch: *::StartPage, TPrintDC::EndPage*

Geschütztes Datenelement

DOCINFO

```
DOCINFO DocInfo;
```

Enthält die Eingabe- und Ausgabe-Dateinamen, die *TPrintDC::StartDoc()* benutzt. Die Struktur DOCINFO definiert sich wie folgt:

```
typedef struct   {
  int cbSize;                  //Größe der Struktur, Bytes
  LPSTR lpszDocName;           //Dokumentname <= 32 Zeichen inkl.
abschließender 0
  LPSTR lpszOutput;            //Name der Ausgabedatei
} DOCINFO;
```

Durch das *lpszOutput*-Feld kann ein Druckauftrag in eine Datei umgeleitet werden. Wenn dieses Feld NULL ist, geht die Ausgabe an das Gerät für das festgelegte DC.

Siehe auch: *TPrintDC::StartDoc*

Struktur TPrintDialog::TData — printdia.h

Die Klasse *TPrintDialog* enthält die notwendigen Informationen zur Initialisierung des Dialogfensters mit der Druckerauswahl des Benutzers. Diese Informationen bestehen aus der Zahl der zu druckenden Exemplare, der ersten und letzten zu druckenden Seite, der höchsten und niedrigsten Seitenzahl, die gedruckt werden kann, sowie verschiedenen Flag-Werten, die anzeigen, ob der Aktionsschalter Seiten angezeigt werden soll, ob das Markierungsfeld Druckausgabe in Datei umleiten aktiviert ist, usw. Diese Struktur wird von *TPrintDialog* für das Initialisieren des Druck-Dialogfensters verwendet. Sie wird immer dann aktualisiert, wenn der Benutzer die Druckanforderungen ändert.

Beim Auftreten eines Fehlers gibt *TPrintDialog::TData* einen der erweiterten allgemeinen Dialog-Fehlercodes zurück. *TPrintDialog::TData* ist auch für das Sperren und Freigeben von Speicher für die Strukturen DEVMODE und DEVNAMES zuständig, die Informationen über den Druckertreiber, den Drucker und den Druckerport enthalten.

TPrinter hat über sein Datenelement *Data* Zugriff auf diese Informationen.

Öffentliche Datenelemente

Copies

```
int copies;
```

Copies gibt die Anzahl der Exemplare an, die gedruckt werden sollen.

Error

```
uint32 Error;
```

Error enthält einen oder mehrere der folgenden *CommDlgExtendedError*-Codes:

Konstante	Bedeutung
CDERR_DIALOGFAILURE	Dialogfenster konnte nicht erstellt werden.
CDERR_FINDRESFAILURE	Angegebene Ressource wurde nicht gefunden.
CDERR_LOCKRESOURCEFAILURE	Angegebene Ressource konnte nicht gesperrt werden.
CDERR_LOADRESFAILURE	Angegebene Ressource konnte nicht geladen werden.
CDERR_MEMALLOCFAILURE	Zuweisung von Speicher für interne Datenstrukturen nicht möglich.
CDERR_MEMLOCKFAILURE	Sperren des einem Handle zugehörigen Speichers nicht möglich.
PDERR_CREATEICFAILURE	*TPrintDialog* konnte keinen Informationskontext herstellen.
PDERR_DEFAULTDIFFERENT	Der durch Strukturelemente beschriebene Drucker entspricht nicht dem Standarddrucker. Diese Fehlermeldung kann auftreten, wenn der Benutzer den im Kontroll-Bedienfeld angegebenen Drucker ändert.
PDERR_DNDMMISMATCH	Der in *DevMode* und *DevNames* angegebene Drucker ist unterschiedlich.
PDERR_GETDEVMODEFAIL	Der Drucker-Gerätetreiber konnte die Struktur *DevMode* nicht initialisieren.
PDERR_INITFAILURE	Die *TPrintDialog*-Struktur konnte nicht initialisiert werden.
PDERR_LOADDRVFAILURE	Der angegebene Drucker-Gerätetreiber konnte nicht geladen werden.
PDERR_NODEFAULTPRN	Ein Standarddrucker konnte nicht gefunden werden.
PDERR_NODEVICES	Keine Druckertreiber vorhanden.
PDERR_PARSEFAILURE	Der String im [Geräte]-Abschnitt der WIN.INI-Datei konnte nicht zerlegt werden.
PDERR_PRINTERNOTFOUND	Der [Geräte]-Abschnitt der WIN.INI-Datei enthält nicht den angegebenen Drucker.
PDERR_RETDEFFAILURE	*DevMode* oder *DevNames* enthält Null.

Flags

```
uint32 Flags;
```

Flags kann aus einer oder mehreren der folgenden Windows-API-Konstanten bestehen:

Konstante	Beschreibung
PD_ALLPAGES	Gibt an, daß das Schaltfeld "Alles" beim Schließen des Dialogfensters angewählt war.
PD_COLLATE	Bewirkt, daß das Markierungsfeld "Sortieren" beim Erstellen des Dialogfensters überprüft wird.

PD_DISABLEPRINTTOFILE	Deaktiviert das Markierungsfeld "In Datei drucken".
PD_HIDEPRINTTOFILE	Blendet das Markierungsfeld "In Datei drucken" aus und deaktiviert es.
PD_NOPAGENUMS	Deaktiviert das Schaltfeld "Seiten" und das betreffende Editierfeld.
PD_NOSELECTION	Deaktiviert das Schaltfeld "Auswahl".
PD_NOWARNING	Verhindert, daß bei fehlendem Standarddrucker eine Warnmeldung eingeblendet wird.
PD_PAGENUMS	Wählt das Schaltfeld "Seiten" beim Erstellen des Dialogfensters.
PD_PRINTSETUP	Blendet anstelle des Dialogfensters "Drucken" das Dialogfenster "Einstellungen Drucker" ein.
PD_PRINTTOFILE	Überprüft beim Erstellen des Dialogfensters das Markierungsfeld "In Datei drucken".
PD_RETURNDC	Gibt einen Gerätekontext zurück, der zu der Benutzerauswahl des Dialogfensters paßt.
PD_RETURNDEFAULT	Gibt *DevNames*-Strukturen ohne Einblenden eines Dialogfensters zurück, die für den Standarddrucker initialisiert werden.
PD_RETURNIC	Gibt einen Informationskontext zurück, der zu der Benutzerauswahl des Dialogfensters paßt.
PD_SELECTION	Wählt das Schaltfeld »Auswahl« beim Erstellen des Dialogfensters.
PD_SHOWHELP	Zeigt das Hilfe-Feld im Dialogfenster.
PD_USEDEVMODECOPIES	Unterstützt ein Druckertreiber Mehrfachkopien, dann bewirkt das Setzen dieses Flags, daß die gewünschte Kopienanzahl im *dmCopies*-Element der *DevMode*-Struktur und 1 in *Copies* gespeichert wird. Werden Mehrfachkopien durch den Druckertreiber nicht unterstützt, wird durch das Setzen dieses Flags das Editierfeld "Kopien" deaktiviert. Wird das Flag nicht gesetzt, wird die Zahl 1 in *DevMode* und die gewünschte Kopienanzahl in *Copies* gespeichert.

FromPage

```
int FromPage;
```

FromPage gibt die erste zu druckende Seite an.

MaxPage

```
int MaxPage;
```

MaxPage gibt den Maximalwert für die zu druckenden Anfangs- und Endseiten an.

MinPage

```
int MinPage;
```

MinPage gibt den Minimalwert für die zu druckenden Anfangs- und Endseiten an.

ToPage

```
int ToPage;
```

ToPage gibt die zu druckende Endseite an.

Siehe auch: *::PRINTDLG*

Öffentliche Elementfunktionen

ClearDevMode

```
void ClearDevMode();
```

Hebt die Gerätemodus-Information auf (zur Initialisierung der Dialogelemente erforderliche Information).

ClearDevNames

```
void ClearDevNames();
```

Hebt die Gerätename-Information auf (Information, die drei Strings zur Angabe von Treibername, Druckername und Ausgabe-Schnittstelle enthält).

GetDeviceName

```
const char far* GetDeviceName() const;
```

Fragt den Namen der Ausgabeeinheit ab.

GetDevMode

```
const DEVMODE far* GetDevMode() const;
```

Holt einen Zeiger auf eine DEVMODE-Struktur (eine Struktur, die die zur Initialisierung der Dialogelemente erforderliche Information enthält).

GetDevNames

```
const DEVNAMES far* GetDevNames() const;
```

Holt einen Zeiger auf eine DEVNAMES-Struktur (Information, die drei Strings zur Angabe von Treibername, Druckername und Ausgabe-Schnittstelle enthält).

GetDriverName

```
const char far* GetDriverName() const;
```

Fragt den Namen des Drucker-Gerätetreibers ab.

GetOutputName

```
const char far* GetOutputName() const;
```

Fragt den Namen des physikalischen Ausgabemediums ab.

Lock

```
void Lock();
```

Sperrt den zu DEVMODE und DEVNAMES gehörenden Speicher.

SetDevMode

```
void SetDevMode(const DEVMODE far* devMode);
```

Setzt die Werte für die DEVMODE-Struktur.

SetDevNames

```
void SetDevNames(const char far* driver, const char far* device,
                 const char far* output);
```

Setzt die Werte für die DEVNAMES-Struktur.

TransferDC

```
TPrintDC* TransferDC();
```

Erstellt ein *TPrintDC* und gibt es mit den aktuellen Einstellungen zurück.

Unlock

```
void Unlock();
```

Hebt die Sperre des zu DEVMODE und DEVNAMES gehörenden Speichers auf.

Klasse TPrintDialog — printdia.h

TPrintDialog zeigt entweder ein modales Drucker-Dialogfenster oder ein Druckerkonfigurations-Fenster an. In dem Drucker-Dialogfenster werden die Merkmale eines bestimmten Druckauftrags festgelegt. Das Drucker-

konfigurations-Dialogfenster erlaubt das Ändern der Drucker-Einstellungen. Zusätzliche Merkmale für den Druckauftrag können festgelegt werden.

Öffentlicher Konstruktor

Konstuktor

```
TPrintDialog(TWindow* parent, TData& data, const char far*
             printTemplateName=0, const char far* setupTemplateName=0,
             const char far* title=0, TModule* module=0);
```

Erzeugt ein Druck- oder Druckereinstellungs-Dialogfenster mit festgelegten Daten, übergeordnetem Fenster, Fenstertitel und Modul.

Öffentliche Elementfunktionen

DoExecute

```
int DoExecute();
```

DoExecute kopiert Flags und Druckbeschreibungen nach *data*, wenn kein Fehler auftritt. Wenn ein Fehler auftritt, setzt *DoExecute* die Fehlernummer *data* auf einen *CommDlgExtendedError*-Code.

GetDefaultPrinter

```
bool GetDefaultPrinter();
```

GetDefaultPrinter erkennt den Gerätemodus und den Namen, die für den System-Standarddrucker initialisiert wurden, ohne Anzeige eines Dialogfensters.

Geschützte Datenelemente

Data

```
TData& Data;
```

Data ist eine Referenz zu dem *TData*-Objekt, das im Konstruktor übergeben wird.

Siehe auch: *TPrintDialog::TData*

pd

```
PRINTDLG pd;
```

Legt die Merkmale des Dialogfenster-Druckauftrags wie Seitenbereich, Anzahl der Kopien, Gerätekontext usw. fest, die zum Initialisieren des Drucks oder des Druckereinstellungs-Dialogfensters erforderlich sind. ObjectWindows übergibt diese Information an die Windows API *PrintDlg*-Funktion.

Siehe auch: *TPrintDialog::TData*

Geschützte Elementfunktionen

CmSetup

```
void CmSetup();
```

Reagiert auf den Klick des Einstellungsschalters mit einer EV_COMMAND-Botschaft.

DialogFunction

```
bool DialogFunction(uint message, WPARAM, LPARAM);
```

Gibt **true** zurück, wenn eine Botschaft behandelt wird.

Siehe auch: *TDialog::DialogFunction*

Antworttabelleneinträge

Die *TPrintDialog*-Antworttabelle besitzt keine Einträge.

Klasse TPrinter — printer.h

TPrinter stellt das physikalische Druckausgabegerät dar. Eine Instanz von *TPrinter* muß initialisiert werden, um zu drucken oder um einen Drucker zu konfigurieren.

Öffentlicher Konstruktor und Destruktor

Konstruktor

```
TPrinter();
```

Erzeugt eine *TPrinter*-Instanz, die sich auf den Standarddrucker bezieht. Um den Drucker zu wechseln, muß *SetDevice* nach Initialisierung des Objektes aufgerufen werden, oder *Setup* wird aufgerufen, damit der Benutzer über ein Dialogfenster das neue Gerät wählen kann.

Destruktor

```
virtual ~TPrinter();
```

Gibt die Ressourcen, die *TPrinter* zugewiesen wurden, frei.

Öffentliche Elementfunktionen

ClearDevice

```
void ClearDevice();
```

Wird durch *SetPrinter* und den Destruktor aufgerufen. *ClearDevice* hebt die Zuordnung des Geräts mit dem aktuellen Drucker auf. *ClearDevice* än-

dert den aktuellen Status des Druckers auf PF_UNASSOCIATED. Dies bewirkt, daß das Objekt alle Aufrufe von *Print* ignoriert, bis das Objekt wieder einem Drucker zugeordnet wird.

Print

```
bool Print(TWindow* parent, TPrintout& printout, bool prompt);
```

Print übergibt das bezeichnete und zu druckende Objekt auf das zugehörigen Druckausgabegerät und blendet während des Drucks ein Abbruch-Dialogfenster ein. Außerdem werden alle während des Druckens auftretenden Fehler angezeigt.

Siehe auch: *TPrinter::Error*

ReportError

```
virtual void ReportError(TWindow* parent, TPrintout& printout);
```

Print ruft *ReportError* auf, wenn Fehler auftreten. Standardmäßig erscheint das System-Meldungsfenster mit einem Fehler-String, der aus der Standard-Stringtabelle erstellt wurde. Diese Funktion kann umgangen werden, um ein benutzerdefiniertes Fehler-Dialogfenster anzuzeigen.

Setup

```
void Setup(TWindow* parent);
```

Diese Funktion sollte aufgerufen werden, wenn der Anwender den aktuell verbundenen Drucker auswählen und/oder konfigurieren soll. *Setup* ruft nochmals das Dialogfenster Druckereinstellungen auf, das dem Benutzer angezeigt wird.

Geschützte Datenelemente

Data

```
TPrintDialog::TData* Data;
```

Data ist eine Referenz auf die *TPrintDialog*-Datenstruktur, die Informationen über die Druckerauswahl des Benutzers enthält.

Siehe auch: *TPrintDialog::TData* struct

Error

```
int Error;
```

Error ist der Fehlercode, den das GDI während des Druckens zurückgibt. Dieser Wert wird durch einen Aufruf von *Print* initialisiert.

Geschützte Elementfunktionen

CreateAbortWindow

```
virtual TWindow* CreateAbortWindow(TWindow* parent, TPrintout& printout);
```

Erzeugt ein Druckabbruch-Dialogfenster.

ExecPrintDialog

```
virtual bool ExecPrintDialog(TWindow* parent);
```

Führt einen *TPrintDialog* aus.

GetDefaultPrinter

```
void GetDefaultPrinter();
```

Aktualisiert die Druckerstruktur mit Informationen über den Standarddrucker des Anwenders.

SetPrinter

```
void SetPrinter(const char* driver, const char* device,
                const char* output);
```

SetPrinter ändert die Verbindung des Geräts Drucker. *Setup* ruft *SetPrinter* auf, um die Verbindung interaktiv zu ändern. Die für diese Methode gültigen Parameter sind in dem [devices]-Abschnitt der WIN.INI-Datei enthalten.

Einträge im [devices]-Abschnitt besitzen das folgende Format:

```
<device name >=<driver>, <port> {,<port>}
```

Klasse TPrinter::TXPrinter — printer.h

Die verschachtelte Klasse *TXPrinter* beschreibt eine Exception, die aus einem ungültigen Druckerobjekt resultiert. Dieser Fehler kann bei der Druckausgabe auf einem physikalischen Drucker auftreten.

Öffentlicher Konstruktor

Konstruktor

```
TXPrinter(uint resId = IDS_PRINTERERROR);
```

Konstruiert ein TXPrinter-Objekt mit der Standardbotschaft IDS_PRINTERERROR.

Klasse TPrinterAbortDlg — printer.h

TPrinterAbortDlg ist der Objekttyp des Standard-Dialogfensters zum Druckabbruch. Dieses Dialogfenster wird initialisiert, um den Titel des aktuellen Ausdrucks einzublenden wie auch das Gerät und die Schnittstelle, über die gegenwärtig gedruckt wird.

TPrinterAbortDlg erwartet drei statische Textelemente mit Steuerelement-ID 101 für den Titel, 102 für das Gerät und 103 für die Schnittstelle. Diese Elemente müssen "%s" an beliebiger Stelle im Text-String enthalten, damit sie durch Titel, Gerät und Schnittstelle ersetzt werden können. Die

Dialogfenster-Elemente können jede beliebige Position und Tabulatorreihenfolge aufweisen.

Öffentliche Konstruktoren

Konstruktor

```
TPrinterAbortDlg(TWindow* parent, TResId resId, const char far* title,
                 const char far* device, const char far* port);
```

Konstruiert ein Abbruch-Dialogfenster, das eine Schaltfläche Abbrechen enthält und die Angaben zum Titel, dem Gerät und der Schnittstelle anzeigt.

Öffentliche Elementfunktionen

EvCommand

```
virtual LRESULT EvCommand(uint id, HWND hWndCtl, uint notifyCode);
```

Verwaltet den Abbruch-Schalter des Dialogfensters zum Druckabbruch.

SetupWindow

```
void SetupWindow();
```

Verbindet Objekte mit der Dialogfenster-Ressourcenvorlage, damit Titel, Gerät und Schnittstelle für den Druck festgelegt werden können. Vergleiche die Erläuterungen zu *TPrintoutFlags* für Informationen zum Drucken von Flags und Druckerstatus-Informationen.

Siehe auch: *TPrintoutFlags* enum

Klasse TPrintout — printer.h

TPrintout stellt das physikalische gedruckte Dokument dar, das an einen Drucker zum Ausdruck geschickt werden soll. *TPrintout* übernimmt die Übergabe des Dokuments an den Drucker. Da dieser Objekttyp abstrakt ist, kann er allein nicht benutzt werden, um etwas zu drucken. Für jedes Dokument oder jeden Dokumenttyp muß eine von *TPrintout* abgeleitete Klasse erstellt werden, und die *PrintPage*-Funktion muß überschrieben werden.

Öffentlicher Konstruktor und Destruktor

Konstruktor

```
TPrintout(const char far* title);
```

Erzeugt eine Instanz von *TPrintout* mit dem angegebenen Titel.

Destruktor

```
virtual ~TPrintout();
```

Zerstört die durch den Konstruktor zugewiesenen Ressourcen.

Öffentliche Elementfunktionen

BeginDocument

```
virtual void BeginDocument(int startPage, int endPage, unsigned flags);
```

Die *Print*-Funktion des Druckerobjekts ruft vor dem Druck jeder Kopie eines Dokuments einmal *BeginDocument* auf. Das *flags*-Feld gibt an, ob der aktuelle Druckabschnitt Grafik, Text oder beides akzeptiert.

Die Standardeinstellung *BeginDocument* hat keine Wirkung. Abgeleitete Objekte können *BeginDocument* überschreiben, um alle notwendigen Initialisierungen am Beginn jeder Kopie des Dokuments durchzuführen.

Siehe auch: *TPrintoutFlags* enum

BeginPrinting

```
virtual void BeginPrinting();
```

Die *Print*-Funktion des Druckerobjekts ruft zu Beginn eines Druckauftrags einmal *BeginPrinting* auf, und zwar unabhängig davon, wie viele Kopien des Dokuments gedruckt werden sollen. Abgeleitete Objekte können *BeginPrinting* überschreiben, um alle notwendigen Initialisierungen vor Beginn des Drucks durchzuführen.

EndDocument

```
virtual void EndDocument();
```

Die *Print*-Funktion des Druckerobjekts ruft nach Beenden des Drucks jeder Kopie des Dokuments *EndDocument* auf. Abgeleitete Objekte können *EndDocument* überschreiben, um alle notwendigen Schritte am Ende jedes Dokuments durchzuführen.

EndPrinting

```
virtual void EndPrinting();
```

Die *Print*-Funktion des Druckerobjekts ruft nach Beenden des Drucks aller Kopien des Dokuments *EndPrinting* auf. Abgeleitete Objekte können *EndPrinting* außer Kraft setzen, um alle notwendigen Schritte am Ende jedes Dokuments durchzuführen.

GetDialogInfo

```
virtual void GetDialogInfo(int& minPage, int& maxPage, int& selFromPage,
                           int& selToPage);
```

GetDialogInfo liefert die benötigte Information, um ausgewählte Seiten des Dokuments drucken zu können. **true** wird zurückgegeben, wenn eine Seitenauswahl möglich ist. Die Verwendung von Seitenbereichen (von ... bis) ist optional. Wenn jedoch die Seitenzahlen ohne Schwierigkeiten bestimmt werden können, setzt *GetDialogInfo* die Anzahl der Seiten des Dokuments fest. Sonst sollte die Seitenzahl auf 0 gesetzt werden. Der Druck wird dann fortgesetzt, bis *HasPage* **false** zurückgibt.

GetTitle

```
const char far* GetTitle() const;
```

Liefert den Titel des aktuellen Druckjobs zurück.

HasPage

```
virtual bool HasPage(int pageNumber);
```

HasPage wird nach dem Druck jeder Seite aufgerufen. Standardmäßig gibt es **false** zurück. Dies bedeutet, daß nur eine Seite gedruckt werden soll. Wenn das Dokument mehr als eine Seite enthält, muß diese Funktion außer Kraft gesetzt werden, um **true** zurückzugeben, solange noch mehr Seiten zu drucken sind.

PrintPage

```
virtual void PrintPage(int page, TRect& rect, unsigned flags);
```

PrintPage wird für jede Seite (oder für jeden Abschnitt, wenn *Banding* **true** ist) aufgerufen und muß außer Kraft gesetzt werden, um den Inhalt der gewünschten Seite zu drucken. Die *rect*- und *flag*-Parameter werden für die abschnittsweise Bearbeitung benutzt, um den Umfang und den Typ des Abschnitts anzugeben, der vom Treiber aktuell angefordert wird (sollte ignoriert werden, wenn *Banding* **false** ist). *page* bezeichnet die Seitennummer der aktuellen Seite.

SetPrintParams

```
virtual void SetPrintParams(TPrintDC* dc, TSize pageSize);
```

SetPrintParams setzt *DC* auf *dc* und *PageSize* auf *pageSize*. Die *Print*-Funktion des Druckerobjekts ruft *SetPrintParams* auf, um die benötigte Information zur Bestimmung der Paginierung und des Seitenzählers zu erhalten. Abgeleitete Objekte, die *SetPrintParams* außer Kraft setzen, müssen die geerbte Funktion aufrufen.

Siehe auch: *TPreviewPage::Paint*
@MARGINAL = WantBanding

```
bool WantBanding() const;
```

Gibt den Wert des Datenelements *Banding* zurück.

WantForceAllBands

```
bool WantForceAllBands() const;
```

Liefert den Wert des Datenelements *ForceAllBands* zurück.

Typdefinition

TPrintoutFlags enum

```
enum{pfGraphics, pfText, pfBoth};
```

ObjectWindows definiert die folgenden Konstanten für abschnittweise Bearbeitung, die zum Setzen von Flags für Druckobjekte verwendet werden.

Konstante	Beschreibung
pfBoth	Aktueller Abschnitt akzeptiert sowohl Text als auch Grafik.

pfGraphics	Aktueller Abschnitt akzeptiert nur Grafik.
pfText	Aktueller Abschnitt akzeptiert nur Text.

Siehe auch: *TPrinter, TPrintOut*

Geschützte Datenelemente

Banding

```
bool Banding;
```

Wenn *Banding* **true** ist, wird der Ausdruck abschnittsweise bearbeitet, und die *PrintPage*-Funktion wird einmal für jeden Abschnitt aufgerufen. Andernfalls wird *PrintPage* nur einmal pro Seite aufgerufen. Die abschnittsweise Bearbeitung eines Ausdrucks nutzt die Zeit und den Speicher besser. Standardmäßig ist *Banding* auf **false** gesetzt.

DC

```
TPRintDC* DC;
```

DC ist das Handle für den Gerätekontext, der zum Drucken benutzt werden muß.

ForceAllBands

```
bool ForceAllBands;
```

Viele Gerätetreiber stellen nicht alle Druckerabschnitte zur Verfügung, wenn nicht sowohl Text als auch Grafik im ersten Abschnitt (der typischerweise ein Abschnitt ist, der nur Text enthält) verarbeitet werden. Wird *ForceAllBands* auf **true** belassen, wird der Druckertreiber gezwungen, alle Abschnitte unabhängig von der Art der Aufrufe in der *PrintPage*-Funktion zur Verfügung zu stellen. Wenn *PrintPage* nur reine Textdarstellung bewirkt, ist *ForceAllBands* **false** die effizientere Einstellung. Die Standardeinstellung ist **true**. *ForceAllBands* wirkt sich nur aus, wenn *Banding* **true** ist.

PageSize

```
TSize PageSize;
```

PageSize ist die Größe des Druckbereichs auf dem Ausdruck der Seite.

Title

```
char far* Title;
```

Title ist der aktuelle Titel, der für den Ausdruck benutzt werden soll. Standardmäßig erscheint dieser Titel im Abbruch-Dialogfenster wie auch als Name des Druckauftrags im Druck-Manager.

Klasse TPrintPreviewDC — preview.h

TPrintPreviewDC ist von *TPrintDC* abgeleitet und bildet die Drucker-Gerätekoordinaten auf logische Bildschirm-Koordinaten ab. Es setzt den Umfang des Sichtfensters fest und bestimmt die Bildschirm- und Drucker-Schriftattribute. Viele der *TPrintPreviewDC*-Funktionen haben Priorität vor den virtuellen Funktionen von *TDC*.

Öffentlicher Konstruktor und Destruktor

Konstruktor

```
PrintPreview(TDC& screen, TPrintDC& printdc, const TRect& client, const
             TRect& clip);
```

Der Konstruktor des *TPrintPreviewDC* nimmt einen Bildschirm-Gerätekontext wie auch einen Drucker-Gerätekontext. Der Bildschirm-Gerätekontext wird an den geerbten Konstruktor übergeben, und der Drucker-Gerätekontext wird in das Element *PrnDC* kopiert.

Destruktor

```
~TPrintPreviewDC();
```

Zerstört ein *TPrintDC*-Objekt.

Öffentliche Elementfunktionen

GetDeviceCaps

```
int GetDeviceCaps(int index) const;
```

GetDeviceCaps gibt Informationen über die Fähigkeiten des Drucker-Gerätekontexts - wie z.B. Schrift- und Pitch-Attribute - zurück. Das *index*-Argument gibt den benötigten Informationstyp an.

Siehe auch: *TDC::GetDeviceCaps*

LPtoSDP

```
bool LPtoSDP(TPoint* points, int count = 1) const;
```

Wandelt jeden der *count*-Punkte im *points*-Array von logischen Punkten des Drucker-Gerätekontexts in Bildschirmpunkte um. Ist der Aufruf erfolgreich, wird ein Wert ungleich Null zurückgegeben. Andernfalls wird 0 zurückgegeben.

LPtoSDP

```
bool LPtoSDP(TRect& rect) const;
```

Wandelt jeden der Punkte im Parameter *rect* von logischen Punkten des Drucker-Gerätekontexts in Bildschirmpunkte um. Ist der Aufruf erfolgreich, wird ein Wert ungleich Null zurückgegeben. Andernfalls wird 0 zurückgegeben.

Siehe auch: *TPrintPreviewDC::SDPtoLP, TDC::LPtoDP*

OffsetViewportOrg

```
bool OffsetViewportOrg(const TPoint& delta, TPoint far* oldOrg = 0);
```

Modifiziert den Ursprung des Gerätekontext-Grafikfensters relativ zu den aktuellen Werten. Die *delta x*- und *y*-Komponenten werden zum vorhergehenden Ursprung addiert, und der daraus resultierende Punkt wird der neue Ursprung des Grafikfensters. Der alte Ursprung wird in *oldOrg* gespeichert. Ist der Aufruf erfolgreich, wird ein Wert ungleich Null zurückgegeben. Andernfalls wird 0 zurückgegeben.

Siehe auch: *TPrintPreviewDC::SetViewportOrg, TDC::OffsetViewportOrg*

ReOrg

```
virtual void ReOrg();
```

Holt die x- und y-Ausmaße des Grafikfensters, gleicht die logischen und die Bildschirmpunkte an und setzt die x- und y-Ausmaße des Grafikfensters zurück.

ReScale

```
virtual void ReScale();
```

Bildet die Punkte des Drucker-Gerätekontexts auf dem Bildschirm-DC ab. Setzt das Bildschirmfenster-Ausmaß gleich dem größten logischen Zeiger des Drucker-Gerätekontexts.

RestoreFont

```
void RestoreFont();
```

Stellt das ursprüngliche GDI-Schriftobjekt für diesen Gerätekontext wieder her.

Siehe auch: *TPrintPreviewDC::SelectObject, TDC::OrgFont*

ScaleViewportExt

```
bool ScaleViewportExt(int xNum, int xDenom, int yNum, int yDenom,
                      TSize far* oldExtent = 0);
```

Modifiziert die Ausmaße des Gerätekontext-Grafikfensters relativ zu den aktuellen Werten. Die neuen Ausmaße werden wie folgt abgeleitet:

xNewVE=(xOldVE**xNum)/ xDenom*
yNewVE=(yOldVE**yNum)/ yDenom*

Die alten Ausmaße werden in *oldExtent* gespeichert. Ist der Aufruf erfolgreich, wird ein Wert ungleich Null zurückgegeben. Andernfalls wird 0 zurückgegeben.

Siehe auch: *TDC::ScaleViewportExt, TPrintPreviewDC::SetViewportExt*

ScaleWindowExt

```
bool ScaleWindowExt(int xNum, int xDenom, int yNum, int yDenom,
                    TSize far* oldExtent = 0);
```

Modifiziert die Ausmaße des Gerätekontext-Fensters relativ zu den aktuellen Werten. Die neuen Ausmaße werden wie folgt abgeleitet:

xNewWE=(xOldWE**xNum)/ xDenom*
yNewWE=(yOldWE**yNum)/ yDenom*

Die alten Ausmaße werden in *oldExtent* gespeichert. Ist der Aufruf erfolgreich, wird ein Wert ungleich Null zurückgegeben. Andernfalls wird 0 zurückgegeben.

Siehe auch: *TDC::SetWindowExt, TPrintPreviewDC::ScaleWindowExt*

SDPtoLP

```
bool SDPtoLP(TPoint* points, int count = 1) const;
```

Wandelt jeden der *count*-Punkte im *points*-Array von Bildschirmgerät-Punkten in logische Punkte des Drucker-Gerätekontexts um. Ist der Aufruf erfolgreich, wird ein Wert ungleich Null zurückgegeben. Andernfalls wird 0 zurückgegeben.

SDPtoLP

```
bool SDPtoLP(TRect& rect) const;
```

Wandelt jeden der Punkte im Parameter *rect* von Bildschirmgerät-Punkten in logische Punkte des Drucker-Gerätekontexts um. Ist der Aufruf erfolgreich, wird ein Wert ungleich Null zurückgegeben. Andernfalls wird 0 zurückgegeben.

Siehe auch: *TPrintPreviewDC::LPtoSDP, TDC::DPtoLP*

SelectObject

```
void SelectObject(const TFont& newFont);
```

Wählt das vorgegebene Schriftobjekt für den Gerätekontext aus.

Siehe auch: *TPrintPreviewDC::SelectStockObject, TDC::SelectObject*

SelectStockObject

```
void SelectStockObject(int index);
```

Liefert ein Handle zu der vordefinierten Schrift.

Siehe auch: *TDC::SelectStockObject*

SetBkColor

```
TColor SetBkColor(TColor color);
```

Setzt die aktuelle Hintergrundfarbe des Gerätekontexts auf den angegebenen *color*-Wert oder auf den ihm nächstliegenden verfügbaren Wert. Gibt 0x80000000 zurück, wenn der Aufruf scheitert.

Siehe auch: *TDC::SetBkColor*

SetMapMode

```
int SetMapMode(int mode);
```

Setzt den aktuellen Mapping-Modus des Gerätekontexts auf *mode*. Gibt den Wert des vorangegangenen Mapping-Modus zurück. Der Mapping-Modus legt fest, wie logische Koordinaten auf Gerätekoordinaten abgebildet werden. Er steuert außerdem die Ausrichtung der x- und y-Achsen des Geräts.

Siehe auch: *TDC::GetMapMode, TDC::SetMapMode*

SetTextColor

```
TColor(TColor color);
```

Setzt die aktuelle Textfarbe des Gerätekontexts auf den angegebenen *color*-Wert. Die Textfarbe bestimmt die Farbe, die durch *TDC::TextOut* und *TDC::EndTextOut* angezeigt wird.

Siehe auch: *TDC::GetTextColor, TDC::SetTextColor*

SetViewportExt

```
bool SetViewportExt(const TSize& extent, TSize far* oldExtent = 0);
```

Setzt die x- und y-Ausmaße des Grafikfensters des Bildschirms auf die angegebenen *extent*-Werte. Die alten Ausmaße werden in *oldExtent* gespeichert. Ist der Aufruf erfolgreich, wird ein Wert ungleich Null zurückgegeben. Andernfalls wird 0 zurückgegeben. Der *extent*-Wert legt fest, welches Maß an Dehnung oder Stauchung im logischen Koordinatensystem notwendig ist, um sich dem Geräte-Koordinatensystem anzupassen. *extent* legt außerdem die relative Ausrichtung der zwei Koordinatensysteme fest.

Siehe auch: *TDC::GetViewportExt, TDC::SetViewportExt*

SetViewportOrg

```
bool OffsetViewportOrg(const TPoint& delta, TPoint far* oldOrg = 0);
```

Ändert den Ursprung des Grafikfensters des Drucker-Gerätekontexts gemäß den aktuellen Werten. Die x- und y-Komponenten aus delta werden zu dem vorhergehenden Ursprung addiert, und der errechnete Punkt wird zum neuen Ursprung des Grafikfensters. Der alte Ursprung wird in *oldOrg* gespeichert. Ist der Aufruf erfolgreich, wird ein Wert ungleich Null zurückgegeben. Andernfalls wird 0 zurückgegeben.

Siehe auch: *TPrintPreviewDC::OffsetViewportOrg, TDC::GetViewportOrg, TDC::SetViewportOrg*

SetWindowExt

```
bool SetWindowExt(const TSize& extent, TSize far* oldExtent = 0);
```

Setzt die x- und y-Ausmaße des Fensters des Gerätekontexts auf die angegebenen *extent*-Werte. Die alten Ausmaße werden in *oldExtent* gespeichert. Ist der Aufruf erfolgreich, wird ein Wert ungleich Null zurückgegeben. Andernfalls wird 0 zurückgegeben. Der *extent*-Wert legt fest, welches Maß an Dehnung oder Stauchung im logischen Koordinatensystem notwendig ist, um sich dem Geräte-Koordinatensystem anzupassen. *extent* legt außerdem die relative Ausrichtung der zwei Koordinatensysteme fest.

Siehe auch: *TDC::GetWindowExt, TDC::SetWindowExt, TPrintPreviewDC::ScaleWindowExt*

SyncFont

```
virtual void SyncFont();
```

Setzt die Bildschirmschrift gleich der aktuellen Druckerschrift.

Geschützte Datenelemente

CurrentPreviewFont

```
TFont* CurrentPreviewFont;
```

Die aktuelle Schriftart der Darstellung.

PrnDC

```
TPrintDC& PrnDC;
```

Enthält eine Referenz zu dem Drucker-Gerätekontext.

PrnFont

```
HFONT PrnFont;
```

Die aktuelle Druckerschriftart.

Geschützte Elementfunktionen

GetAttributeHDC

```
HDC GetAttributeHDC() const;
```

Gibt die Attribute des Drucker-Gerätekontexts (*PrnDC*) zurück.

Siehe auch: *TDC::GetAttributeHDC*

Klasse TProfile profile.h

Eine Instanz von *TProfile* kapselt eine Einstellung in einer Systemdatei, oft auch Profil- oder Initialisierungsdatei bezeichnet. Beispiele für diesen Dateityp stellen die Windows-Initialisierungsdateien SYSTEM.INI und WIN.INI dar. Innerhalb der Systemdatei sind die einzelnen Einstellungen abschnittsweise zusammengefaßt, wie zum Beispiel

[Diagnostics]; Abschnittsname
Enabled=0 ; Einstellung

Bei Einstellungen heißt der Wert links vom Gleichheitszeichen *Schlüssel*. Der Wert rechts vom Gleichheitszeichen, der eigentliche *Wert*, kann entweder eine Ganzzahl oder eine Zeichenkette sein.

Öffentlicher Konstruktor und Destruktor

Konstruktor

```
TProfile(const char* section, const char* filename=0);
```

Konstruiert ein Objekt *TProfile* für den angegebenen Abschnitt *section* innerhalb der Profildatei *filename*. Wird kein Dateiname angegeben, ist automatisch die Profildatei des Systems gesetzt, z.B. WIN.INI unter Windows.

Destruktor

```
~TProfile();
```

Entfernt das *TProfile*-Objekt.

Öffentliche Elementfunktionen

GetInt

```
int GetInt(const char* key, int defaultInt = 0);
```

Schlägt den mit der angegebenen Zeichenkette *key* verbundenen Ganzzahlwert nach und liefert ihn zurück. Wird *key* nicht gefunden, wird der Standardwert *default key* zurückgegben.

GetString

```
bool GetString(const char* key, char buff[], unsigned buffSize, const
char* defaultString = "");
```

Schlägt den mit der angegebenen Zeichenkette *key* verbundenen String-Wert nach und liefert ihn zurück. Der Stringwert wird bis zu einer Größe von *buffSize* Bytes in *buff* kopiert. Wird der String nicht gefunden, wird der in *defaultString* enthaltene Standardwert zurückgegeben. Wird ein Schlüssel 0 übergeben, werden alle Abschnittwerte in *buff* zurückgeliefert.

WriteInt

```
boolWriteInt(const char* key, const char* int value);
```

Schlägt den Schlüssel nach und ersetzt seinen Wert durch den Ganzzahlwert, der in (*int*) übergeben wird. Wenn der Schlüssel nicht gefunden wird, nimmt *WriteInt* einen neuen Eintrag vor. Gibt bei Erfolg **true** zurück.

WriteString

```
bool WriteString(const char* key, const char* str);
```

Schlägt den Schlüssel nach und ersetzt seinen Wert mit dem in *str* übergebenen String-Wert. Wird der Schlüssel nicht gefunden, macht *WriteString* einen neuen Eintrag. Gibt bei Erfolg **true** zurück.

Klasse TPXPictureValidator validate.h

TPXPictureValidator-Objekte vergleichen die Eingaben des Benutzers mit der Maske eines Datenformats, um die Gültigkeit der eingegebenen Daten zu überprüfen. Die Masken sind kompatibel mit den von Borlands relationaler Datenbank Paradox zur Kontrolle der Dateneingabe benutzten Masken. Vergleiche zur vollständigen Beschreibung der Masken-Bezeichner die Elementfunktion *Picture*.

Öffentliche Konstruktoren

Konstruktor

```
TPXPictureValidator(const char far* pic, bool autoFill=false);
```

Erzeugt ein Masken-Validierungsobjekt durch Aufruf des von *TValidator* geerbten Konstruktors. *pic* wird so gesetzt, daß es darauf zeigt. Dann wird das *voFill*-Bit in *Options* gesetzt, wenn *AutoFill* **true** ist, und *Options* wird auf *voOnAppend* gesetzt. Sollte die Maske ungültig sein, wird eine *TXValidator*-Exception erzeugt.

Öffentliche Elementfunktionen

Error

```
void Error();
```

Überschreibt die virtuelle Funktion von *TValidator* und blendet ein Meldungsfenster ein, das einen Fehler im Maskenformat anzeigt, sowie den String einblendet, auf den *pic* zeigt.

Siehe auch: *TValidator::Error*

IsValid

```
bool IsValid(const char far* str);
```

IsValid überschreibt die virtuelle Funktion von*TValidato*r und vergleicht den in *str* übergebenen String mit der Formatmaske, die in *Pic* festgelegt wurde. *IsValid* gibt **true** zurück, wenn *Pic* NULL ist oder wenn *Picture Complete* für *str* zurückgibt, das heißt, wenn *str* keine weiteren Eingaben benötigt, um dem angegebenen Format zu genügen. Andernfalls wird **false** zurückgegeben.

Siehe auch: *TPXPictureValidator::Picture*

IsValidInput

```
bool IsValidInput(char far* str, bool suppressFill);
```

IsValidInput überschreibt die virtuelle Funktion von *TValidator* und vergleicht den in *str* übergebenen String mit der Formatmaske, die in *Pic* fest-

gelegt wurde. *IsValid* gibt **true** zurück, wenn *Pic* NULL ist oder wenn *Picture* nicht *Error* für *str* zurückgibt. Andernfalls wird **false** zurückgegeben. Der *suppressFill*-Parameter setzt für die Dauer des Aufrufs von *IsValidInput* den Wert in *voFill* außer Kraft.

Wenn *suppressFill* **false** und *voFill* gesetzt ist, gibt der Aufruf von *Picture* einen gefüllten, auf *str* basierenden String zurück, so daß das Bild im Editierfeld automatisch das in *Pic* festgelegte Format wiedergibt.

Siehe auch: *TPXPictureValidator::Picture*

Picture

```
virtual TPicResult Picture(char far* input, bool autoFill=false);
```

Formatiert den in *input* übergebenen String gemäß dem Format, das durch den Masken-String, auf den *Pic* zeigt, festgelegt wurde. *Picture* gibt *prError* zurück, wenn ein Fehler im Masken-String auftritt oder wenn *input* Daten enthält, die zu der angegebenen Maske nicht passen. Gibt *prComplete* zurück, wenn *input* die angegebenen Maskenanforderungen vollständig erfüllt. Gibt *prIncomplete* zurück, wenn *input* Daten enthält, die die angegebenen Maskenanforderungen nur unvollständig erfüllen.

Die folgenden Zeichen werden verwendet, um Formatmasken zu erstellen:

Tabelle 2.28: Zeichen Maskenformat

Zeichenart	Zeichen	Beschreibung
Sonderzeichen	#	Nur eine Ziffer akzeptieren
	?	Nur einen Buchstaben akzeptieren (case_insensitive)
	&	Nur einen Buchstaben akzeptieren (Großschreibung erzwingen)
	@	Jedes Zeichen akzeptieren
	!	Jedes Zeichen akzeptieren, Großschreibung erzwingen
Vergleichszeichen	;	Nächstes Zeichen literal behandeln
	*	Wiederholungszähler
	[]	Option
	{}	Gruppen-Operatoren
	'	Alternativsatz
Alle anderen werden literal behandelt		

Siehe auch: *TPicResult* enum

Geschütztes Datenelement

Pic

```
string Pic;
```

Zeigt auf einen String, der die Maske enthält, die das Datenformat im zugehörigen Editierfeld bestimmt. Der Konstruktor setzt *Pic* auf einen String, der als einer der Parameter übergeben wird.

Klasse TRadioButton radiobut.h

TRadioButton ist ein Oberflächenobjekt, das ein entsprechendes Schaltfeld in Windows repräsentiert. *TRadioButton* wird verwendet, um ein Schaltfeld in einem übergeordneten *TWindow* zu erstellen. Ferner kann ein *TRadioButton* dazu verwendet werden, die Kommunikation zwischen Ihrer Anwendung und den Schaltfeldern eines *TDialog*-Objektes zu erleichtern.

Schaltfelder haben zwei Zustände: markiert und unmarkiert. *TRadioButton* erbt Elementfunktionen für die Statusverwaltung von seiner Basisklasse *TCheckBox*. Optional kann ein Schaltfeld Teil einer Gruppe (*TGroupBox*) sein, in der die Schaltfelder visuell und logisch zusammengefaßt werden. *TRadioButton* ist eine streamfähige Klasse.

Öffentliche Konstruktoren

Konstruktor

```
TRadioButton(TWindow* parent, int id, const char far* title, int x,
             int y, int w, int h, TGroupBox *group, TModule* module = 0);
```

Erstellt ein Schaltfeldobjekt mit dem angegebenen übergeordneten Fenster (*parent*), der Dialogelement-ID (*id*), dem zugehörigen Text (*title*), der Position (*x*,*y*) im Verhältnis zum Ursprung des Client-Bereiches des übergeordneten Fensters, mit der Breite (*w*), der Höhe (*h*) und dem zugehörigen Gruppenfeld (*group*). Ruft den *TCheckBox*-Konstruktor mit den entsprechenden Parametern auf und setzt das Datenelement *Attr.Style* auf WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON.

Siehe auch *TControl::TControl*

```
TRadioButton(TWindow* parent, int resourceId, TGroupBox, *group,
             TModule* module = 0);
```

Erstellt ein *TRadioButton*-Objekt, das mit dem Schaltfeld eines *TDialog* verknüpft werden soll. Ruft den *TCheckBox*-Konstruktor mit identischen Parametern auf. Der Parameter *resourceId* muß einer von Ihnen definierten Schaltfeldressource entsprechen.

Geschützte Elementfunktionen

BNClicked

```
void BNclicked();
```

Beantwortet eine ankommende BN_CLICKED-Botschaft.

GetClassName

```
char far* GetClassName();
```

Gibt »BUTTON« zurück. Dies ist der Name der vordefinierten BWCC- oder Windows-Schaltfeldklasse

Antworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_MESSAGE (BM_SETSTYLE, BMSetStyle)	BMSetStyle
EV_WM_GETDlgCODE	EvGetDlgCode

Klasse TRangeValidator — validate.h

Ein *TRangeValidator*-Objekt entscheidet, ob die vom Benutzer eingegebenen Daten in einen festgelegten Integerbereich fallen. *TRangeValidator* ist eine streamfähige Klasse.

Öffentliche Konstruktoren

Konstruktor

```
TRangeValidator (long min, long max);
```

Erstellt ein Bereichsprüfungsobjekt, indem zunächst der von *TFilterValidator* geerbte Konstruktor aufgerufen wird, der einen die Ziffern '0'...'9' und die Zeichen '+' und '-' enthaltenden Zeichensatz übergibt. Setzt *Min* auf *min* und *Max* auf *max* und richtet so den Bereich von Long-Integer-Werten ein, die akzeptiert werden.

Siehe auch *TFilterValidator::TFilterValidator*

Öffentliche Elementfunktionen

Error

```
void Error();
```

Error überschreibt die virtuelle Funktion von *TValidator* und zeigt ein Meldungsfenster mit der Mitteilung an, daß der eingegebene Wert nicht in den spezifizierten Bereich fällt.

IsValid

```
bool IsValid(const char far* str);
```

Wandelt den String *str* in eine Integerzahl um und liefert **true** zurück, wenn das Ergebnis jede der drei folgenden Bedingungen erfüllt:

- Es ist ein zulässiger Integer
- Sein Wert ist größer oder gleich *min*
- Sein Wert ist kleiner oder gleich *max*

Schlägt einer dieser Tests fehl, so gibt IsValid **false** zurück.

Transfer

```
uint Transfer(char far* str, void* buffer, TTransferDirection direction);
```

Enthält die drei Typen *tdSizeData*, *tdGetData* und *tdSetData*, die ein Bereichsprüfer für sein zugehöriges Editierfeld verarbeiten kann. *str* ist der Stringwert des Editierfeldes und *buffer* sind die Daten, die an das Editierfeld übergeben werden. Abhängig vom Wert von *direction* setzt *Transfer* entweder *str* auf den in *buffer* enthaltenen Zahlenwert oder *buffer* auf den Wert des Strings *str*. Entspricht *direction tdSetData*, so bestimmt *Transfer str* aus *buffer*. Entspricht *direction tdGetData*, bestimmt *Transfer buffer* aus *str*. Entspricht *direction tdSizeData*, bestimmt die *Transfer*-Funktion weder Daten noch liest sie diese.

Transfer gibt stets die Größe der übertragenen Daten zurück.

Siehe auch *TWindow::Transfer*

Geschützte Datenelemente

Max

```
long Max;
```

Max ist der höchste gültige Long-Integer-Wert für das Editierfeld.

```
long Min;
```

Min ist der niedrigste gültige Wert eines Long-Integer für das Editierfeld.

Klasse TRegion gdiobjec.h

TRegion, von *TGdiobject* abgeleitet, repräsentiert abstrakte Formen oder Bereiche der GDI (grafische Geräteschnittstelle). *TRegion* kann Bereichsobjekte in unterschiedlichen Formen erstellen. Für das Kombinieren und Vergleichen von Bereichen stehen verschiedene Operatoren zur Verfügung.

Öffentliche Datenelemente

enum TEllipse

```
enum TEllipse{Ellipse};
```

Definiert die klassenspezifische Konstante *Ellipse*, die dazu verwendet wird, den Ellipsenkonstruktor vom Rechteck-Kopierkonstruktor zu unterscheiden.

Siehe auch *TRegion::(const TRect& rect), TRegion::TRegion (const TRect& E, TEllipse)*

Öffentliche Konstruktoren

Konstruktor

```
TRegion();
```

Der Standardkonstruktor erzeugt ein leeres *TRegion*-Objekt. *Handle* ist auf 0, *ShouldDelete* auf **true** gesetzt.

Siehe auch *TGdiObject::Handle*, *TGdiObject::ShouldDelete*, *::CreateRectRgn*

Konstruktor

```
TRegion(HRGN handle, TAutoDelete autoDelete = NoAutoDelete);
```

Erzeugt ein *TRegion*-Objekt und setzt die *Handle*-Datenelemente auf das gegebene eingelesene *handle*. Das *ShouldDelete*-Datenelement ist auf **false** voreingestellt und stellt sicher, daß das eingelesene Handle nicht gelöscht wird, wenn das C++ Objekt zerstört wird. *HRGN* ist der das Handle einer abstrakten Form repräsentierende Windows-Datentyp.

Konstruktor

```
TRegion(const TRegion& region);
```

Dieser öffentliche Kopier-Konstruktor erzeugt eine Kopie des gegebenen *TRegion*-Objektes in der Art:

```
TRegion myRegion = yourRegion;
```

Siehe auch *::CreateRectRgn*, *::CombineRgn*

Konstruktor

```
TRegion(const TRect& rect);
```

Erzeugt ein Bereichsobjekt aus dem gegebenen *TRectangle*-Objekt in der Art:

```
TRegion myRegion(rect1);
TRegion* pRegion;
pRegion = new TRegion(rect2);
```

Siehe auch:*CreateRectRgnIndirect*, *TRect*

Konstruktor

```
TRegion(const TRect& E, TEllipse);
```

Erzeugt das elliptische *TRegion*-Objekt, welchem das gegebene Rechteck *E* einbeschrieben ist. Das *TEllipse*-Argument unterscheidet diesen Konstruktor vom *TRegion(const TRect& rect)*-Konstruktor.

Siehe auch:*CreateEllipticRgnIndirect*, *TRect*

Konstruktor

```
TRegion(const TRect& rect, const TSize& corner);
```

Erzeugt ein *TRegion*-Objekt aus dem gegebenen *rect* und aus dem gegebenen *corner*.

Siehe auch *TRect*, *TSize*

Konstruktor

```
TRegion(const TPoint* points, int count, int fillMode);
```

Erzeugt ein gefülltes *TRegion*-Objekt aus den durch *points* und *fillMode* gegebenen Polygonen.

Siehe auch *::CreatePolygonRgn*, *TPoint*

Konstruktor

```
TRegion(const TPoint* points, const int* polyCounts, int count,
        int fillMode);
```

Erzeugt ein gefülltes *TRegion*-Objekt aus den durch *points* und *fillMode* gegebenen Polygonen.

Siehe auch *::CreatePolyPolygonRgn*, *TPoint*

Öffentliche Elementfunktionen

Contains

```
inline bool Contains(const TPoint& point) const;
```

Liefert **true** zurück, wenn der vorliegende Bereich den gegebenen Punkt enthält.

Siehe auch *::PtInRegion*, *TPoint*

GetRgnBox

```
inline int GetRgnBox(TRect& box) const;
inline TRect GetRgnBox() const;
```

Ermittelt das begrenzende Rechteck (das minimale Rechteck, in dem dieser Bereich enthalten ist). In der ersten Version wird das resultierende Rechteck in *box* eingefügt und die zurückgegebenen Werte sind die folgenden:

Wert	Bedeutung
COMPLEXREGION	Bereich hat überlappende Rahmen.
NULLREGION	Bereich ist leer.
SIMPLEREGION	Bereich hat keine überlappenden Rahmen.

In der zweiten Version wird das resultierende Rechteck zurückgeliefert.

Siehe auch *::GetRgnBox*, *TRect*

operator==

```
inline bool operator==(const TRegion& other) const;
```

Liefert **true** zurück, wenn der vorliegende Bereich dem Bereich in *other* entspricht.

Siehe auch *::EqualRgn*, *TRegion::operator!=*

operator!=

```
inline bool operator!=(const TRegion& other) const;
```

Liefert **true** zurück, wenn der vorliegende Bereich nicht dem Bereich in *other* entspricht.

Siehe auch *::EqualRgn*, *TRegion::operator==*

operator=

```
TRegion& operator=(const TRegion& source);
```

Weist den *source*-Bereich für den vorliegenden Bereich zu. Eine Referenz zum Ergebnis wird zurückgeliefert, wobei verkettete Zuweisungen zugelassen werden.

Siehe auch *::CombineRgn*

operator+=

```
TRegion& operator+=(const TSize& delta);
```

Zählt zu jedem Punkt dieses Bereichs das gegebene *delta* hinzu, um ihn um *delta.x* und *delta.y* zu verschieben (zu übertragen). Liefert eine Referenz zum resultierenden Bereich zurück.

Siehe auch *::OffsetRgn*, *TSize*, *TRegion::operator-=*

operator-=

```
TRegion& operator-=(const Tsize& delta);
TRegion& operator-=(const TRegion& source);
```

Die erste Version subtrahiert von jedem Punkt des vorliegenden Bereichs das gegebene *delta*, um ihn um *-delta.x* und *-delta.y* zu verschieben (zu übertragen). Die zweite Version verwendet das RGN_DIF-Argument in *::CombineRegion*, um einen »Differenz-Bereich« zu erzeugen, der all jene Teile des vorliegenden Bereichs enthält, die nicht Teile des *source*-Bereichs sind. Beide Versionen liefern eine Referenz zum resultierenden Bereich zurück.

Siehe auch *::OffsetRgn, ::CombineRgn, TSize, TRegion::operator+=, RGN_DIF*

operator&=

```
TRegion& operator&=(const TRegion& source);
TRegion& operator&=(const TRect& source);
```

Erstellt die Schnittmenge dieses Bereichs mit dem gegebenen *source*-Bereich oder Rechteck und liefert eine Referenz zum Ergebnis zurück.

Siehe auch *::CombineRgn*, *TRect*

operator|=

```
TRegion& operator|=(const TRegion& source);
TRegion& operator|=(const TRect& source);
```

Erzeugt die Vereinigungsmenge dieses Bereichs und des gegebenen *source*-Bereichs oder -Rechtecks und liefert eine Referenz zum Ergebnis zurück.

Siehe auch *::CombineRgn*, *TRect*

operator^=

```
TRegion& operator^=(const TRegion& source);
TRegion& operator^=(const TRect& source);
```

Erzeugt die Vereinigungsmenge dieses Bereichs und des gegebenen source-Bereichs oder -Rechtecks, schließt jedoch alle überlappenden Bereiche aus. Liefert eine Referenz zum resultierenden Bereichsobjekt zurück.

Siehe auch *::CombineRgn*, *TRect*

operator HRGN()

```
inline operator HRGN() const;
```

Typumwandlungsoperator. *HRGN* ist der das Handle eines physischen Bereichs repräsentierende Windows-Datentyp.

SetRectRgn

```
inline void SetRectRgn (const TRect& rect);
```

Verwendet die Win-API- SetRectRgn-Funktion, um ein Rechteck der durch *rect* gegebenen Größe zu erzeugen. (Macht keinen Gebrauch von der lokalen Speicherverwaltung.)

Siehe auch *::SetRectRgn*, *TRect*

Touches

```
inline bool Touches(const TRect& rect) const;
```

Liefert **true** zurück, wenn der vorliegende Bereich das gegebende Rechteck berührt.

Siehe auch *::RectInRegion*, *TRect*

Aufzählung TRelationship — layoutco.h

```
enum TRelationship;
```

TRelationship wird von der *TLayoutConstraint*-Struktur verwendet und bestimmt das Verhältnis zwischen den Kanten und Größen eines Fensters und den Kanten und Größen eines anderen Fensters (das über- oder gleichgeordnet sein kann). Diese Verhältnisse können entweder bestimmt werden als gleichwertig zum über- oder gleichgeordneten Fenster (*ImAsIs*), als absoluter Wert (*ImAbsolute*), als Prozentanteil eines der Fenster (*ImPercentOf*) oder als ein Wert, der von unten (*ImBelow*) oder rechts (*ImRightOf*) von einem der Fenster abgezogen wird.

Siehe auch *TLayoutConstraint struct*

Klasse TReplaceDialog — findrepl.h

TReplaceDialog erzeugt ein nicht-modales Dialogfenster, das den Benutzer eine Auswahl von zu ersetzendem Text eingeben läßt.

Öffentliche Konstruktoren

Konstruktor

```
TReplaceDialog (TWindow* parent, TData& data, TResID templateName=0,
                const char far* title=0, TModule* module=0);
```

Erzeugt ein *TReplaceDialog*-Objekt mit einem übergeordneten Fenster, einer Ressourcen-ID und einem Titel. Setzt die Attribute des Dialogfensters mit Hilfe der festgelegten Daten.

Siehe auch *TFindReplaceDialog::TData*

Geschützte Elementfunktionen

DoCreate

```
HWND DoCreate();
```

Erzeugt ein nicht-modales Dialogfenster.

Siehe auch *TDialog::DoCreate*

Klasse TResponseTableEntry eventhan.h

TResponseTableEntry ist eine Template-Klasse, mit der Sie eine Eingabemaske für eine Antworttabelle erstellen können. Die Einträge bestehen aus einer Botschaft, einem Benachrichtigungscode, einer Ressourcen-ID, einem Sendetyp und einem Zeiger auf eine Elementfunktion. Für eine Erläuterung des Formats für Antworttabelleneinträge siehe Kapitel 3.

Für zusätzliche Informationen über die Makros in den Antworttabellen siehe DECLARE_RESPONSE_TABLE und DEFINE_RESPONSE_TABLE.

Öffentliche Datenelemente

Dispatcher

```
TAnyDispatcher Dispatcher;
```

Dispatcher ist ein abstrakter Sendetyp, der auf eine der Dispatcherfunktionen zeigt.

Id

```
uint Id;
```

Enthält das Menü oder das Ressourcen-ID-Kürzel (CM_xxxx) für die Kommandobotschafts-Elementfunktion.

Msg

```
uint Msg;
```

Enthält die ID der gesendeten Botschaft. Es können dies Kommandobotschaften, Botschaften mit der ID untergeordneter Fenster, benachrichtigungsbasierte Botschaften wie LBN_SELCHANGE oder Windows-Botschaften wie LBUTTONDOWN sein.

NotifyCode

```
uint NotifyCode;
```

Speichert den Kontrollbenachrichtigungscode (z.B. ID_LISTBOX) für den Antworttabelleneintrag. Es können dies ein Schaltfeld, ein Kombinationsfenster, ein Editierfeld oder Benachrichtigungscodes für Listen sein.

Pmf

```
PMF Pmf;
```

Pmf zeigt auf das aktuelle Handle oder die aktuelle Elementfunktion.

T

```
typedef void(T::*PMF)();
```

Typ einer generischen Elementfunktion, die auf Benachrichtigungsbotschaften reagiert. *T* ist die Vorlage für die Antworttabelle.

Klasse TRgbQuad color.h

TRgbQuad ist eine aus der Struktur *tagRGBQUAD* abgeleitete Supportklasse, die wie folgt definiert ist:

```
typedef struct tagRGBQUAD {
    BYTE  rgbBlue;
    BYTE  rgbGreen;
    BYTE  rgbRed;
    BYTE  rgbReserved;
} RGBQUAD;
```

wobei *rgbBlue*, *rgbGreen* und *rgbRed* die relative Blau-, Grün- und Rotintensität einer Farbe festlegen. *rgbReserved* wird nicht verwendet und muß auf 0 gesetzt werden.

TRgbQuad wird in Verbindung mit den Klassen *TPalette* und *TColor* verwendet, um RGBQUAD-basierte Farboptionen zu vereinfachen. Es stehen Konstruktoren zur Erstellung von *TRgbQuad*-Objekten aus expliziten RGB-Werten, *TColor*-Objekten oder anderen *TRgbQuad*-Objekten zur Verfügung.

Öffentliche Konstruktoren

Konstruktor

```
TRgbQuad(int r, int g, int b);
```

Erzeugt ein *TRgbQuad*-Objekt mit jeweils auf *r*, *g* und *b* gesetzten *rgbRed*, *rgbGreen* und *rgbBlue*. *rgbReserved* ist auf 0 gesetzt.

Siehe auch *tagRGBQUAD*-Struktur

Konstruktor

```
TRgbQuad(TColor c);
```

Erzeugt ein *TRgbQuad*-Objekt mit jeweils auf *c.Red()*, *c.Green()* und *c.Blue* gesetzten *rgbRed*, *rgbGreen* und *rgbBlue*. *rgbReserved* ist auf 0 gesetzt.

Siehe auch *tagRGBQUAD*-Struktur, *TColor::Red*, *TColor::Green*, *TColor::Blue*

Konstruktor

```
TRgbQuad(const RGBQUAD far& q);
```

Erzeugt ein *TRgbQuad*-Objekt mit denselben Werten wie das referenzierte RGBQUAD-Objekt

Siehe auch *tagRGBQUAD*-Struktur

Klasse TRgbTriple color.h

TRgbTriple ist eine aus der Struktur *tagRgbTriple* abgeleitete Supportklasse, die folgendermaßen definiert ist:

```
typedef struct tagRGBTRIPLE {
    BYTE  rgbBlue;
    BYTE  rgbGreen;
    BYTE  rgbRed;
} RGBTRIPLE;
```

wobei *rgbBlue*, *rgbGreen* und *rgbRed* die relative Blau-, Grün- und Rotintensität einer Farbe bestimmen.

TRgbTriple wird in Verbindung mit den Klassen *TPalette* und *TColor* verwendet, um bmci-color-basierte Operationen zu vereinfachen. Es stehen Konstruktoren zur Erstellung von *TRgbTriple*-Objekten aus expliziten RGB-Werten, *TColor*-Objekten oder anderen *TRgbTriple*-Objekten zur Verfügung.

Öffentliche Konstruktoren

Konstruktor

```
TRgbTriple(int r, int g, int b);
```

Erzeugt ein *TRgbTriple*-Objekt mit jeweils auf *r*, *g* und *b* gesetzten *rgbRed*, *rgbGreen* und *rgbBlue*.

Siehe auch *tagRGBTRIPLE*-Struktur

Konstruktor

```
TRgbTriple(TColor c);
```

Erzeugt ein *TRgbTriple*-Objekt mit jeweils auf *c.Red()*, *c.Green()* und *c.Blue()* gesetzten *rgbRed*, *rgbGreen* und *rgbBlue*. *rgbReserved* ist auf 0 gesetzt.

Siehe auch *tagRGBTRIPLE*-Struktur, *TColor::Red*, *TColor::Green*, *TColor::Blue*

Konstruktor

```
TRgbTriple(const RGBTRIPLE far& t);
```

Erzeugt ein *TRgbTriple*-Objekt mit denselben Werten wie das bezogene RGBTRIPLE-Objekt

Siehe auch *tagRGBTRIPLE*-Struktur

Klasse TScreenDC dc.h

TScreenDC ist eine aus *TWindowDC* abgeleitete Klasse, die direkten Zugriff auf das Bildschirm-Bitmap erlaubt. *TScreenDC* erhält ein DC für ein Window Handle 0, das ohne Clipping für den gesamten Bildschirm gilt. Window Handle 0 zeichnet oberhalb anderer Fenster.

Öffentliche Konstruktoren

Konstruktor

```
TScreenDC();
```

Voreingestellter Konstruktor für *TScreenDC*-Objekte.

Klasse TScrollBar scrollba.h

TScrollBar-Objekte repräsentieren selbständige vertikale und horizontale Bildlaufleisten. Die meisten *TScrollBar*-Elementfunktionen verwalten Position und Bereich des Positionszeigers der Bildlaufleiste.

Eine besondere Eigenschaft des Typs *TScrollBar* ist der auf Benachrichtigungen basierende Satz von Elementfunktionen, die die Stellung des Positionszeigers automatisch als Antwort auf Bildlaufleisten-Botschaften von Windows anpassen.

TScrollBar-Objekte dürfen niemals in Fenstern plaziert werden, die unter ihren Attributen die Stile WS_HSCROLL oder WS_VSCROLL haben.

TScrollBar ist eine streamfähige Klasse.

Öffentliche Datenelemente

LineMagnitude

```
int LineMagnitude;
```

LineMagnitude ist die Größe der Schrittweite, um die der Bildlauf durchgeführt werden soll, wenn der Anwender durch Anklicken der Pfeile in den Bildlaufleisten eine kleinere Verschiebung wünscht. Der *TScrollBar*-Konstruktor setzt *LineMagnitude* per Voreinstellung auf 1. (Die Voreinstellung für den Bildlaufbereich liegt zwischen 0 und 100.)

Siehe auch *TScrollBar::SetupWindow*

PageMagnitude

```
int PageMagnitude;
```

PageMagnitude ist die Größe der Schrittweite, um die der Bildlauf durchgeführt werden soll, wenn der Anwender durch Anklicken der Pfeile in den Bildlaufleisten eine größere Verschiebung wünscht. Der *TScrollBar*-Konstruktor setzt *PageMagnitude* per Voreinstellung auf 10. (Der Bildlaufbereich liegt standardmäßig zwischen 0 und 100.)

Öffentliche Konstruktoren

Konstruktor

```
TScrollBar(TWindow* parent, int id, int x, int y, int w, int h, bool
           isHScrollBar, TModule* module = 0);
```

Konstruiert und initialisiert ein *TScrollBar*-Objekt mit dem gegebenen übergeordneten Fenster (*parent*), einer Dialogelement-ID (*id*), einer Positi-

on (*x*, *y*) sowie der Breite (*w*) und Höhe (*h*). Ruft den *TControl*-Konstruktor mit den entsprechenden Parametern auf. Ist *isHScrollBar* **true**, so fügt die Funktion zu den in *Attr.Style* angegebenen Stilen SBS_HORZ hinzu. Wenn *isHScrollBar* **false** ist, wird SBS_VERT eingefügt. Wenn die angegebene Höhe für eine horizontale Bildlaufleiste oder die angegebene Breite für eine vertikale Bildlaufleiste 0 beträgt, wird der Standardwert verwendet. *LineMagnitude* wird mit 1 initialisiert und *PageMagnitude* auf 10 gesetzt.

Siehe auch *TControl::TControl*

Konstruktor

```
TScrollBar(TWindow* parent, int resourceId, TModule* module = 0);
```

Erzeugt ein *TScrollBar*-Objekt, das mit einem Bildlaufleisten-Dialogelement eines *TDialog*-Objektes verknüpft werden soll. Ruft den *TControl*-Konstruktor mit identischen Parametern auf.

Der Parameter *resourceId* muß einer von Ihnen zu definierenden Bildlaufleistenressource entsprechen.

Öffentliche Elementfunktionen

DeltaPos

```
virtual int DeltaPos(int delta);
```

Ruft *SetPosition* auf, um die Stellung des Positionszeigers der Bildlaufleiste um den in *delta* angegebenen Wert zu verändern. Ein positiver Wert in *delta* verschiebt den Positionszeiger nach unten oder rechts, ein negativer *delta*-Wert nach oben oder links. Die neue Zeigerposition wird zurückgeliefert.

Siehe auch *TScrollBar::SetPosition*

EvHScroll

```
void EvHScroll(uint scrollCode, uint thumbPos, HWND hWndCtl);
```

Antworttabellen-Behandlungsroutine, die eine virtuelle Funktion (*SBBottom*, *SBLineDown* usw.) in Antwort auf Botschaften aufruft, welche von *TWindow::DispatchScroll* aus gesendet wurden.

EvVScroll

```
void EvVScroll(uint scrollCode, uint thumbPos, HWND hWndCtl);
```

Antworttabellen-Behandlungsroutine, die eine virtuelle Funktion (*SBBottom*, *SBLineDown* usw.) in Antwort auf Botschaften aufruft, welche von *TWindow::DispatchScroll* aus gesendet wurden.

GetPosition

```
virtual int GetPosition() const;
```

Liefert die aktuelle Position des Positionszeigers zurück.

Siehe auch *TScrollBar::SetPosition*, *TScrollBarData*-Struktur

GetRange

```
virtual void GetRange(int& min, int& max) const;
```

Liefert in *min* und *max* die Grenzen des aktuellen Bildlaufbereichs zurück.

Siehe auch *TScrollBar::SetPosition*, *TScrollBar::SetRange*, *TScrollBarData*-Struktur

SBBottom

```
virtual void SBBottom();
```

Ruft *SetPosition* auf, um den Positionszeiger an das untere Ende oder die rechte Seite der Bildlaufleiste zu verschieben. *SB_BOTTOM* wird aufgerufen, wenn der Positionszeiger auf die unterste oder die am weitesten rechts liegende Position der Bildlaufleiste gezogen wird.

Siehe auch *TScrollBar::SetPosition*

SBLineDown

```
virtual void SBLineDown();
```

Ruft *SetPosition* auf, um den Positionszeiger (in *LineMagnitude*-Einheiten) nach unten oder rechts zu bewegen. *SBLineDown* wird als Antwort auf das Anklicken des unteren oder rechten Pfeils der Bildlaufleiste aufgerufen.

Siehe auch *TScrollBar::SetPosition*

SBLineUp

```
virtual void SBLineUp();
```

Ruft *SetPosition* auf, um den Positionszeiger (in *LineMagnitude*-Einheiten) nach oben oder links zu bewegen. *SBLineUp* wird als Antwort auf das Anklicken des oberen oder linken Pfeils der Bildlaufleiste aufgerufen.

Siehe auch *TScrollBar::SetPosition*

SBPageDown

```
virtual void SBPageDown();
```

Ruft *SetPosition* auf, um den Positionszeiger (*in PageMagnitude*-Einheiten) nach unten oder rechts zu bewegen. *SBPageDown* wird als Antwort auf das Anklicken des unteren oder rechten Bildlaufbereichs der Bildlaufleiste aufgerufen.

Siehe auch *TScrollBar::SetPosition*

SBPageUp

```
virtual void SBPageUp();
```

Ruft *SetPosition* auf, um den Positionszeiger (*in PageMagnitude*-Einheiten) nach oben oder links zu bewegen. *SBPageUp* wird als Antwort auf das Anklicken des oberen oder linken Bildlaufbereichs der Bildlaufleiste aufgerufen.

Siehe auch *TScrollBar::SetPosition*

SBThumbPosition

```
virtual void SBThumbPosition(int thumbPos);
```

Ruft *SetPosition* auf, um den Positionszeiger zu bewegen. *SBThumbPosition* wird aufgerufen, wenn der Positionszeiger an eine neue Position gebracht werden soll.

Siehe auch *TScrollBar::SetPosition*

SBThumbTrack

```
virtual void SBThumbTrack(int thumbPos);
```

Ruft *SetPosition* auf, um den Positionszeiger zu bewegen, während dieser an eine neue Position gezogen wird.

Siehe auch *TScrollBar::SetPosition*

SBTop

```
virtual void SBTop();
```

Verschiebt den Positionszeiger durch Aufruf von *SetPosition* an den Beginn oder die rechte Seite der Bildlaufleiste. *SBTop* wird aufgerufen, wenn der Positionszeiger auf die Anfangsposition oder die am weitesten rechts gelegene Position der Bildlaufleiste gezogen wird.

Siehe auch *TScrollBar::SetPosition*

SetPosition

```
virtual void SetPosition(int thumbPos);
```

Bewegt den Positionszeiger in die in *ThumbPos* angegebene Stellung. Liegt *ThumbPos* außerhalb des aktuellen Bildlaufbereichs, wird der Positionszeiger an die dazu nächstgelegene Position innerhalb des Bereichs gebracht.

Siehe auch *TScrollBar::GetPosition*

SetRange

```
virtual void SetRange(int min, int max);
```

Setzt die Grenzen des Bildlaufbereichs auf *min* und *max*.

Siehe auch *TScrollBar::GetRange*

Transfer

```
uint Transfer(void* buffer, TTransferDirection direction);
```

Überträgt Bildlaufleisten-Daten in den oder aus dem Transferpuffer, auf den *buffer* zeigt. *buffer* sollte auf eine *TScrollBarData*-Struktur zeigen.

Es werden Daten in oder aus dem Transferpuffer übertragen, wenn *direction* auf *tdGetData* oder *tdSetData* gesetzt ist.

Transfer liefert immer den Umfang der zu übertragenden Daten (die Größe der *TScrollBarData*-Struktur) zurück. Um die Größe dieser Daten zu ermitteln, ohne daß dabei Daten übertragen werden, setzen Sie *tdSizeData* auf *direction*.

Siehe auch *TScrollBarData*-Struktur

Geschützte Elementfunktionen

GetClassName

```
char far* GetClassName();
```

Liefert "SCROLLBAR", den Namen der Windows-Registrierungsklasse von *TScrollBar*, zurück.

SetupWindow

```
void SetupWindow();
```

Setzt den Bildlaufbereich auf 0, 100. Rufen Sie *SetRange* auf, um diesen Bereich neu zu definieren.

Siehe auch *TScrollBar::SetRange*

Antworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_WM_HSCROLL	EvHScroll
EV_WM_VSCROLL	EvVScroll

Struktur TScrollBarData — scrollba.h

Die *TScrollBarData*-Struktur enthält Integer-Werte, die den Bereich von Stellungen des Positionszeigers auf der Bildlaufleiste repräsentieren. Die *TScrollBar*-Funktion *GetRange* ruft *TScrollBarData* auf, um die obersten und untersten Zeigerpositionen auf der Bildlaufleiste zu erfragen. *GetPosition* ruft *TScrollBarData* auf, um die aktuelle Stellung des Positionszeigers auf der Bildlaufleiste zu erfragen.

Siehe auch *TScrollBar::Transfer*

Öffentliche Datenelemente

HighValue

```
int HighValue;
```

Enthält den höchsten Wert für die Stellung des Positionszeigers innerhalb des Bildlaufbereichs der Bildlaufleiste.

Siehe auch *TScrollBar::GetRange*

LowValue

```
int LowValue;
```

Enthält den niedrigsten Wert für die Stellung des Positionszeigers innerhalb des Bildlaufbereichs der Bildlaufleiste.

Siehe auch *TScrollBar::GetRange*

Position

```
int Position;
```

Enthält die Position des Positionszeigers auf der Bildlaufleiste.

Siehe auch *TScrollBar::GetPosition*

Klasse TScroller scroller.h

TScroller unterstützt einen automatischen Bildlaufmechanismus (hier als »*Autoscrolling*« bezeichnet), der in Verbindung mit horizontalen oder vertikalen Bildlaufleisten arbeitet (auch ohne daß Bildlaufleisten vorhanden sind). Ist das Autoscrolling aktiviert, wird ein automatischer Bildlauf durchgeführt, sobald die Maus aus dem Client-Bereich des Fensters herausgezogen wird. Ist das *AutoMode*-Datenelement **true**, führt *TScroller* Autoscrolling durch.

Um *TScroller* zu verwenden, setzen Sie das *Scroller*-Datenelement Ihrer von *TWindow* abgeleiteten Klasse auf ***ein*** *TScroller*-Objekt, das im *TWindow*-Konstruktor dieser abgeleiteten Klasse instantiiert wird. *TScroller* ist eine streamfähige Klasse.

Öffentliche Datenelemente

AutoMode

```
bool AutoMode;
```

Ist **true**, wenn der automatische Bildlauf aktiviert ist.

AutoOrg

```
bool AutoOrg;
```

Ist **true**, wenn der Ursprungspunkt auf der Bildlaufleiste aktualisiert wird.

HasHScrollBar, HasVScrollbar

```
bool HasHScrollBar, HasVSCrollBar;
```

Ist **true**, wenn das zugehörige Fenster eine horizontale oder vertikale Bildlaufleiste hat.

TrackMode

```
bool TrackMode;
```

Ist **true**, wenn Trackscrolling (»Nachführen«) aktiviert ist.

Window

```
TWindow* Window;
```

Zeigt auf das Fenster, für das die Bildlaufleiste des Client-Bereichs verwaltet werden soll.

XLine, YLine

```
int XLine, YLine;
```

Bestimmt die Anzahl der logischen Geräteeinheiten pro Zeile, um die das Rechteck in horizontaler (*X*) und vertikaler (*Y*) Richtung verschoben werden soll.

XPage, YPage

```
int XPage, YPage;
```

Bestimmt die Anzahl der logischen Geräteeinheiten pro Seite, um die das Rechteck in horizontaler (*X*) und vertikaler (*Y*) Richtung verschoben werden soll.

XPos, YPos

```
long XPos, YPos;
```

Bestimmt die aktuelle Position des Rechtecks in horizontalen (*XPos*) und vertikalen (*YPos*) Bildlaufeinheiten.

XRange, YRange

```
long XRange, YRange;
```

Bestimmt die Anzahl horizontaler und vertikaler Bildlaufeinheiten.

XUnit, YUnit

```
int XUnit, YUnit;
```

Bestimmt die Schrittweite (in logischen Geräteeinheiten), um die für das Rechteck ein Bildlauf in horizontaler (*X*) und vertikaler (*Y*) Richtung durchgeführt werden soll. Das Rechteck rollt nach rechts, wenn *XUnit* positiv und nach links, wenn *XUnit* negativ ist. Das Rechteck rollt nach unten, wenn *YUnit* positiv und nach oben, wenn *YUnit* negativ ist.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TScroller(TWindow* window, int xUnit, int yUnit, long xRange, long
          yRange);
```

Erstellt ein *TScroller*-Objekt mit *window* als dem zugehörigen Fenster und *xUnit*, *yUnit*, *xRange* und *yRange* als jeweils *xUnit*, *yUnit*, *xRange* and *yRange*. Initialisiert Datenelemente auf ihre voreingestellten Werte. *HasHScrollBar* und *HasVScrollBar* werden gemäß der Bildlaufleisten-Attribute des zugehörigen Fensters gesetzt.

```
virtual ~TScroller();
```

Enfernt ein *TScroller*-Objekt. Setzt den Wert der *Scroller*-Variable des zugehörigen Fensters auf 0.

Öffentliche Elementfunktionen

AutoScroll

```
virtual void AutoScroll();
```

Führt für das zugehörige Fenster einen Bildlauf durch, wenn die Maus aus dem Fensterbereich herausgezogen wird. Richtung und Umfang des Bildlaufs sind von der aktuellen Mausposition abhängig.

BeginView

```
virtual void BeginView(TDC& dc, TRect& rect);
```

Ist *TScroller::AutoOrg* **true** (Standardbedingung), versetzt *BeginView* während einer Zeichenoperation automatisch den Ursprung der logischen Koordinaten des Client-Bereichs um XPos und YPos. Ist *AutoOrg* **false** (z.B., wenn der Scroller größer als 32.767 Einheiten ist), müssen Sie den Ursprung manuell aktualisieren.

EndView

```
virtual void EndView();
```

Aktualisiert den Positionszeiger der Bildlaufleisten im zugehörigen Fenster so, daß sie der Position von *TScroller* entsprechen.

HScroll

```
virtual void HScroll(uint scrollEvent, int thumbPos);
```

Antwortet mit dem Aufruf von *ScrollBy* oder *ScrollTo* auf das angegebene horizontale Bildlaufereignis (*scrollEvent*). Der Typ des Bildlaufereignisses wird durch die entsprechenden Windows-SB_Konstanten gekennzeichnet. *thumbPos* speichert die aktuelle Stellung des Positionszeigers, wenn der Scroller von den Bildlaufereignissen SB_THUMBTRACK und SB_THUMBPOSITION benachrichtigt wird.

IsAutoMode

```
virtual bool IsAutoMode();
```

IsAutoMode ist **true**, wenn der automatische Bildlauf aktiviert ist.

Siehe auch *TScroller::AutoMode*

IsVisibleRect

```
bool IsVisibleRect(long x, long y, int xExt, int yExt);
```

Ist **true**, wenn das Rechteck (*x*, *y*, *xExt* und *yExt*) sichtbar ist.

SetPageSize

```
virtual void SetPageSize();
```

Setzt die *XPage*- und *YPage*-Datenelemente auf die Breite und Höhe (in *XUnits* und *YUnits*) des Client-Bereichs des zugehörigen Fensters.

Siehe auch *TScroller::XPage*, *YPage*, *TScroller::XUnit*, *YUnit*

SetRange

```
virtual void SetRange(long xRange, long yRange);
```

Setzt die Bereiche *xRange* und *yRange* des *TScroller* auf die spezifizierten Parameter. Ruft dann *SetBarRange* auf, um den Bereich der Bildlaufleisten des zugehörigen Fensters zu synchronisieren.

Siehe auch *TScroller::SetSBarRange*

SetSBarRange

```
virtual void SetSBarRange();
```

Setzt den Bereich der Bildlaufleisten des zugehörigen Fensters, so daß er dem Bereich des *TScroller* entspricht.

SetUnits

```
virtual void SetUnits(int xUnit, int yUnit);
```

Setzt die *XUnit*- und *YUnit*-Datenelemente auf *TheXUnit* bzw. auf *TheYUnit*. Aktualisiert *XPage* und *YPage* durch Aufruf von *SetPageSize*.

Siehe auch *TScroller::XPage*, *YPage*, *TScroller::XUnit*, *YUnit*

ScrollBy

```
void ScrollBy(long dx, long dy);
```

Führt einen Bildlauf bis zu der Position durch, die aufgrund der übergebenen Delta-Werte (*dx*, *dy*) errechnet wurde. Eine positive Delta-Position bewegt den Positionszeiger nach unten oder rechts, eine negative nach oben oder links.

ScrollTo

```
virtual void ScrollTo(long x, long y);
```

Rollt das Rechteck zu der in *x* und *y* spezifizierten Position.

SetWindow

```
inline void SetWindow(TWindow* win);
```

Setzt das zugehörige Fenster auf *win*.

VScroll

```
virtual void VScroll(uint scrollEvent, int thumbPos);
```

Antwortet mit dem Aufruf von *ScrollBy* und *ScrollTo* auf das angegebene vertikale Bildlaufereignis (*scrollEvent*). Der Typ des Bildlaufereignisses wird durch die entsprechenden Windows-SB_Konstanten gekennzeichnet. *thumbPos* enthält die aktuelle Stellung des Positionszeigers, wenn der Scroller von den Bildlaufereignissen SB_THUMBTRACK und SB_THUMBPOSITION benachrichtigt wird.

Siehe auch *TScroller::ScrollTo*

XScrollValue

```
int XScrollValue(long rangeUnit);
```

XScrollValue konvertiert einen Bereichswert der horizontalen Bildlaufleiste in einen horizontalen Bildlaufwert.

Siehe auch *TScroller::YScrollValue*

XRangeValue

```
int XRangeValue(int scrollUnit);
```

XRangeValue konvertiert einen Bildlaufwert der horizontalen Bildlaufleiste in einen horizontalen Bereichswert.

Siehe auch *TScroller::YRangeValue*

YScrollValue

```
int YScrollValue(long rangeUnit);
```

YScrollValue konvertiert einen Bereichswert der vertikalen Bildlaufleiste in einen vertikalen Bildlaufwert.

Siehe auch *TScroller::XScrollValue*

YRangeValue

```
int YRangeValue(int scrollUnit);
```

YRangeValue konvertiert einen Bildlaufwert der horizontalen Bildlaufleiste in einen vertikalen Bereichswert.

Siehe auch *TScroller::XRangeValue*

Klasse TSeparatorGadget — gadget.h

TSeparatorGadget ist eine einfache Klasse, die Sie zur Erstellung eines Separators zwischen Gadgets verwenden können. Hierzu müssen Sie die Größe des Separators in SM_CXBORDER- (Breite des Fensterrahmens) und SM_CYBORDER-Einheiten (Höhe des Fensterrahmens) angeben. Die linke und untere Begrenzung des Separators werden nach Aufruf von *GetsystemMetrics* gesetzt. Per Voreinstellung deaktiviert sich der Separa-

tor selbst und stellt die Größenanpassung ab. Beachten Sie, daß der voreingestellte Rahmenstil »keiner« ist.

Siehe auch *TGadget::TBorderStyle enum*

Öffentliche Elementfunktionen

TSeparatorGadget

```
TSeparatorGadget(int size = 6);
```

Size wird sowohl für die Breite als auch für die Höhe des Separators verwendet und wird mit 6 Rahmeneinheiten (Breite oder Höhe eines schmalen Fensterrahmens) initialisiert.

Klasse TSlider — slider.h

TSlider, eine abstrakte, von *TScrollBar* abgeleitete Klasse, definiert in den Grundzügen das Verhalten von Gleitreglern (Dialogelemente, die verwendet werden, um Positionsinformationen ohne Bildlauf zu liefern). Wie Bildlaufleisten haben Gleitregler sowohl Minimum- und Maximumpositionen als auch eine Zeilen- und Seitengröße. Gleitregler können entweder über die Maus oder die Tastatur bewegt werden. Wenn Sie die Maus verwenden, um den Regler zu bewegen, können Sie den Positionszeiger ziehen, den Führungsschlitz beiderseits des Zeigers anklicken, um den Positionszeiger eine bestimmte Spanne weit zu bewegen (*PageMagnitude*) oder das Lineal anklicken, um den Zeiger an eine bestimmte Position des Führungsschlitzes zu bewegen. Die [*Home*]- und [*End*]-Tasten auf der Tastatur bewegen den Positionszeiger in die Minimum- (*Min*) bzw. Maximumposition (*Max*) auf dem Gleitregler.

TSlider können bewirken, daß der Positionszeiger sich automatisch nach der nächstgelegenen Skalenposition ausrichtet (»Snapping«). Ferner können Sie die Skalenabstände (der Zwischenraum zwischen den Linien, die die Haupteinteilung der X- oder Y-Achse markieren) bestimmen.
Das ObjectWindows Programm SLIDER.CPP, befindet sich auf Ihrer Diskette.

Zu spezifischen Details zu horizontalen und vertikalen Gleitreglern schlagen Sie bitte unter den beiden abgeleiteten Klassen *THSlider* und *TVSlider*, zwei abgeleiteten Klassen, nach.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TSlider(TWindow* parent, int id, int X, int Y, int W, int H, TResId
        thumbResId, TModule* module = 0);
```

Erstellt ein Gleitregler-Objekt, wobei *Pos* und *ThumbRgn* auf 0 gesetzt werden, *TicGap* auf *Range* dividiert durch 10, *SlotThick* auf 17, *Snap* auf **true** und *Sliding* auf **false**. Setzt *Attr.W* und *Attr.H* auf die Werte in *X* und *Y*.

Destruktor

```
~TSlider();
```

Entfernt ein *TSlider*-Objekt und löscht *ThumbRgn*.

Öffentliche Elementfunktionen

GetPosition

```
int GetPosition() const;
```

Liefert die aktuelle Position des Gleitreglers zurück. Überlädt die virtuelle Funktion von *TScrollBar*.

Siehe auch *TSlider::SetPosition*

GetRange

```
void GetRange(int &min, int &max) const;
```

Liefert die Endwerte des aktuellen Bereichs von Stellungen des Gleitregler-Positionszeigers in *min* und *max* zurück. Überlädt die virtuelle Funktion von *TScrollBar*.

Siehe auch *TSlider::SetRange*

SetPosition

```
void SetPosition(int thumbPos);
```

Bewegt den Positionszeiger in die in *thumbPos* festgelegte Stellung. Liegt *thumbPos* außerhalb des aktuellen Gleitreglerbereichs, wird der Positionszeiger in die am nächstgelegene Position innerhalb des festgelegten Bereichs bewegt. Überlädt die virtuelle Funktion von *TScrollBar*.

Siehe auch *TSlider::GetPosition*

SetRange

```
void SetRange(int min, int max);
```

Setzt den Gleitregler auf den Bereich zwischen *min* und *max*. Überlädt die virtuellle Funktion von *TScrollBar*.

Siehe auch *TSlider::GetRange*

SetRuler

```
void SetRuler(int ticGap, bool snap = false);
```

Setzt das Lineal des Gleitreglers. Jeder Regler besitzt ein eingebautes Lineal, das zusammen mit dem Gleitregler gezeichnet wird. Das Lineal, das durch Markierungen eingeteilt oder auch nicht unterteilt sein kann, kann so erstellt werden, daß der Positionszeiger automatisch zum Springen an die Skalenmarkierungen gezwungen wird.

Geschützte Elementfunktionen

EvEraseBkgnd

```
bool EvEraseBkgnd(HDC hDC);
```

Antwortet auf eine WM_ERASEBKGND-Botschaft und löscht den Hintergrund des Gleitreglers, wenn sich der Regler verändert. Ruft die virtuellen Funktionen *PaintRuler*, *PaintSlot*, und *PaintThumb* auf, um die einzelnen Teile des Gleitreglers zu zeichnen. Um ein Flimmern zu vermeiden, wird *EvEraseBkgnd* aufgerufen, um den Hintergrund zu löschen, sobald die Zeichnung erscheint.

Siehe auch *TSlider::EvPaint*

EvGetDlgCode

```
uint EvGetDlgCode(MSG far* msg);
```

Antwortet auf eine WM_GETDLGCODE-Botschaft und läßt den Dialog-Verwalter die Antwort auf das Drücken einer DIRECTION- oder TAB-Taste steuern. Faßt Cursor-Bewegungstasten für die Bewegung des Positionszeigers durch die Rückgabe einer DLGC_WANTARROWS-Botschaft zusammen, die anzeigt, daß Richtungstasten gewünscht werden. Der Parameter *msg* gibt die Art der Botschaft an, beispielsweise eine Steuer- oder Befehlsbotschaft, die an den Dialogfenster-Verwalter geschickt wird.

Siehe auch *TButton::EvGetDlgCode, TWindow::DefaultProcessing, DLGC_xxxx* Dialogsteuerungs-Botschaftskonstanten

EvKeyDown

```
void EvKeyDown(uint key, uint repeatCount, uint flags);
```

EvKeyDown überträgt den virtuellen Tastencode in eine Bewegung und bewegt daraufhin den Positionszeiger. *key* zeigt den virtuellen Tastencode der niedergedrückten Taste an, *repeatCount* speichert, wie oft dieselbe Taste gedrückt wurde, und *flags* enthält eine der folgenden Botschaften, die in virtuelle Tastencodes (VK) übersetzt werden:

Wert	Virtueller Tastencode
SB_PAGEUP	VK_PRIOR
SB_PAGEDOWN	VK_NEXT
SB_BOTTOM	VK_END
SB_TOP	VK_HOME
SB_LINEUP	VK_LEFT(entspricht SB_LINELEFT)
SB_LINEUP	VK_UP
SB_LINEDOWN	VK_RIGHT(entspricht SB_LINERIGHT)
SB_LINEDOWN	VK_DOWN

EvKillFocus

```
void EvKillFocus(HWND hWndGetFocus);
```

Als Antwort auf eine an ein Fenster, das den Kontakt zur Tastatur verliert, gesendete WM_KILLFOCUS-Botschaft verbirgt *EvKillFocus* das Caret zunächst und löscht es darauf.

EvLButtonDblClk

```
void EvLButtonDblClk(uint modKeys, TPoint& point);
```

Antwortet auf eine WM_LBUTTONDBLCLK-Botschaft (die ein Doppelklicken der linken Maustaste durch den Benutzer anzeigt) und verwirft dann die Botschaften, so daß die Basisklasse sie nicht erhält.

EvLButtonDown

```
void EvLButtonDown(uint modKeys, TPoint& point);
```

Antwortet auf das Drücken einer Maustaste mit dem Positionieren des Positionszeigers oder damit, daß mit dem Ziehen der Maus begonnen wird. Wird eine Maustaste gedrückt, wenn sich der Mauszeiger über dem Positionszeiger befindet, betritt *EvLButtonDown* den Sliding-Status. Befindet sich der Mauszeiger auf dem Führungsschlitz, blättert *EvLButtonDown* auf- oder abwärts. Befindet sich der Mauszeiger auf dem Lineal, springt *EvLButtonDown* in die bezeichnete Position. *EvLButtonDown* erzeugt einen Bildlaufcode von SB_THUMBPOSITION, SB_LINEUP, SB_LINEDOWN, SB_PAGEUP, SB_PAGEDOWN, SB_THUMBTRACK.

Siehe auch *TSlider::EvLButtonUp*

EvLButtonUp

```
void EvLButtonUp(uint modKeys, TPoint& point);
```

Wird der Mausknopf losgelassen, beendet *EvLButtonUp* das Verschieben des Reglers, das Blättern oder Springen an eine Linealposition.

Siehe auch *TSlider::EvLButtonDown*

EvMouseMove

```
void EvMouseMove(uint modKeys, TPoint& point);
```

Bewegt die Maus an die angegebene Stelle. Wird die Maus gezogen, positioniert *EvMouseMove* den Positionszeiger und sendet eine entsprechende Botschaft an das übergeordnete Fenster.

EvPaint

```
void EvPaint();
```

Füllt den gesamten Gleitregler - Lineal, Führungsschlitz und Positionszeiger. Ruft die virtuellen Funktionen *PaintRuler*, *PaintSlot*, und *PaintThumb* auf, um die einzelnen Komponenten des Gleitreglers zu zeichnen.

Siehe auch *TSlider::EvEraseBkgnd*

EvSetFocus

```
void EvSetFocus(HWND hWndLostFocus);
```

Erzeugt ein blinkendes Caret, um das Zentrum des aktuellen Fensters anzuzeigen.

Siehe auch *TSlider::GetFocus*

EvSize

```
void EvSize(uint sizeType, TSize& size);
```

Berechnet die Größe des Gleitreglers neu, wenn die Fenstergröße geändert wird.

GetBkColor

```
void GetBkColor(TDC& dc);
```

Sendet eine WM_CTLCOLOR-Botschaft an das übergeordnete Fenster und ruft *dc::GetBkColor* auf, um die Hintergrundfarbe für den Gleitregler zu ermitteln.

HitTest

```
virtual int HitTest(TPoint& point) = 0;
```

Erhält Informationen darüber, wo eine gegebene *X*, *Y*-Stelle in den Bereich des Gleitreglers fällt. Der Rückgabewert wird in *scrollCode*s angegeben. Jede der abgeleiteten Klassen führt Vergleiche durch, um einen Bildlaufcode zurückzuliefern.

Siehe auch *TSlider::NotifyParent*

NotifyParent

```
virtual void NotifyParent(int scrollCode, int pos=0) = 0;
```

Sendet eine WS_HSCROLL-oder WS_VSCROLL-Botschaft an das übergeordnete Fenster.

Siehe auch *TVSlider::HitTest*

PaintRuler

virtual void PaintRuler(TDC& dc) = 0;

Zeichnet das Lineal. Vorausgesetzt wird, daß Führungsschlitz oder Positionszeiger das Lineal nicht überlagern.

PaintSlot

```
virtual void PaintSlot(TDC& dc) = 0;
```

Zeichnet den Führungsschlitz, in dem sich der Positionszeiger bewegt.

PaintThumb

```
virtual void PaintThumb(TDC& dc);
```

Zeichnet den eigentlichen Positionszeiger unter Verwendung einer Ressourcen-DIB, die in die aktuellen Systemfarben für Schalter übersetzt wurde und den Führungsschlitz überlagert.

PointToPos

```
virtual int PointToPos(TPoint& point) = 0;
```

Überträgt einen *X*, *Y*-Punkt in eine Positionsangabe in Gleitreglereinheiten.

Siehe auch *TSlider::PosToPoint*

PosToPoint

```
virtual TPoint PosToPoint(int pos) = 0;
```

Überträgt eine Positionsangabe in Gleitreglereinheiten in *X*, *Y*-Koordinaten.

Siehe auch *TSlider::PointToPos*

SetupThumbRgn

```
virtual void SetupThumbRgn();
```

Erzeugt den Bereich, der die Form des Positionszeigers für die vorliegende Gleitreglerklasse definiert. Obwohl der Standardbereich ein einfaches begrenzendes Rechteck ist, kann er jede beliebige Form haben. Während der Positionszeiger des Gleitreglers bewegt wird, dient dieser Bereich zur Überprüfung der Mausposition und zum Aktualisieren der Zeigerposition.

Siehe auch *TSlider::ThumbRgn*

SetupWindow

```
void SetupWindow();
```

Ruft *TScrollBar::SetupWindow* und *SetupThumbRgn* auf, um das Fenster einzurichten.

Siehe auch *TScrollBar::SetupWindow*

SlideThumb

```
virtual void SlideThumb(TDC& dc, int thumbPosl);
```

Verschiebt den Positionszeiger in eine gegebene Stellung und führt die notwendigen Blitting- und Zeichenoperationen durch.

SnapPos

```
int SnapPos(int pos);
```

Beschränkt *Pos* in der Weise, daß es in den Bereich zwischen *Min* und *Max* fällt und führt Snapping durch Runden von *Pos* auf den nächsten *TicGap* durch sofern Snapping eingeschaltet ist.

Siehe auch *TSlider::TicGap*

Geschützte Datenelemente

Bkcolor

```
TColor BkColor;
```

Speichert die Hintergrundfarbe des Gleitreglers.

CaretRect

```
TRect CaretRect;
```

Stellt einen Bezug zur Position des Rechtecks des Cursors her.

Max

```
int Max;
```

Enthält den Maximumwert der Reglerposition.

```
int Min;
```

Enthält den Minimumwert der Reglerposition.

MouseOffset

```
static TSize MouseOffset;
```

MouseOffset zeigt den Offset von der ursprünglichen oberen linken Position des Rechtecks zur der Position, an der die Maustaste gedrückt wurde. Wenn das Rechteck bewegt wird, kann es somit den gleichen Bezug zur Position des Mausklicks behalten.

Pos

```
int Pos;
```

Gibt an, an welcher Stelle des Reglers sich der Positionszeiger befindet.

Siehe auch *TSlider::GetPosition*

Range

```
uint Range;
```

Enthält die Differenz zwischen der oberen und unteren Grenze des Gleitreglers.

SlideDC

```
static TDC* SlideDC;
```

SlideDC wird verwendet, wenn die Maustaste gedrückt und der Positionsanzeiger in Bewegung ist. Es gibt die Bewegung der Maus auf dem DC wieder.

Sliding

```
bool Sliding;
```

Ist **true**, wenn der Positionszeiger in Bewegung ist, andernfalls **false**.

SlotThick

```
int SlotThick;
```

Zeigt die Dicke des Führungsschlitzes an. Die Voreinstellung ist 17.

```
bool Snap;
```

Ist **true**, wenn Snapping aktiviert ist, ansonsten **false**.

ThumbRect

```
TRect ThumbRect;
```

Enthält das den Positionszeiger begrenzende Rechteck.

ThumbResId

```
TResId ThumbResId;
```

ThumbResId ist die Bitmap-Ressourcen-ID für den Positionszeigerknopf.

ThumbRgn

```
TRegion* ThumbRgn;
```

Erzeugt den Bereich, der die Positionsanzeiger-Form für diese Gleitregler-Klasse festlegt. Obwohl der Standardbereich ein einfaches begrenzendes Rechteck ist, ist jede Form möglich. Während der Positionsanzeiger des Gleitreglers bewegt wird, wird dieser Bereich dazu verwendet, die Mausposition zu testen und die Position des Positionsanzeigers zu aktualisieren.

Siehe auch *TSlider::SetupThumbRgn*

TicGap

```
int TicGap;
```

Bestimmt die Größe der Abstände zwischen Skalenmarkierungen in Pixeln.

Antworttabellen

Antworttabelleneintrag	Elementfunktion
EV_WM_ERASEBKGND	EvEraseBkgnd
EV_WM_GETDLGCODE	EvGetDlgCode
EV_WM_KEYDOWN	EvKeyDown
EV_WM_KILLFOCUS	EvKillFocus
EV_WM_LBUTTONDBLCLK	EvLButtonDblClk
EV_WM_LBUTTONDOWN	EvLButtonDown
EV_WM_LBUTTONUP	EvLButtonUp
EV_WM_MOUSEMOVE	EvMouseMove
EV_WM_PAINT	EvPaint

EV_WM_SETFOCUS	EvSetFocus
EV_WM_SIZE	EvSize

Klasse THSlider slider.h

THSlider, von *TSlider* abgeleitet, stellt Implementierungsdetails für horizontale Schieberegler zur Verfügung. Für eine Abbildung eines horizontalen Schiebereglers siehe *TSlider* .

Öffentliche Konstruktoren

Konstruktor

```
THSlider(TWindow* parent, int id, int X, int Y, int W, int H,
         TResId thumbResId, TModule* module = 0);
```

Erzeugt mit einer Standard-Bitmap-Ressourcen-ID von IDB_HSLIDER-THUMB für den Positionsanzeiger-Knopf ein Schieberegler-Objekt.

Geschützte Elementfunktionen

HitTest

```
int HitTest(TPoint& point);
```

Überschreibt die virtuelle Funktion von *TSlider* und holt die Information, an welcher Stelle eine gegebene X, Y-Position auf dem Schieberegler liegt. Der Wert wird in *scrollCodes* zurückgegeben.

Siehe auch *TSlider::HitTest*

NotifyParentNotifyParent

```
void NotifyParent(int scrollCode, int pos=0);
```

Überschreibt die virtuelle Funktion von *TSlider* und sendet eine WS_HSCROLL-Botschaft an das übergeordnete Fenster.

Siehe auch *TSlider::NotifyParent*

PaintRuler

```
void PaintRuler(TDC&);
```

Überschreibt die virtuelle Funktion von *TSlider* und zeichnet die horizontale Skala.

Siehe auch *TSlider::PaintRuler*

PaintSlot

```
void PaintSlot(TDC&);
```

Überschreibt die virtuelle Funktion von *TSlider* und zeichnet den Schlitz des Reglers.

Siehe auch *TSlider::PaintSlot*

PointToPos

```
int PointToPos(TPoint& point);
```

Überschreibt die virtuelle Funktion von *TSlider* und übersetzt einen X,Y-Punkt in eine Positionsangabe in Schieberegler-Einheiten.

Siehe auch *TSlider::PointToPos*

PosToPoint

```
TPoint PosToPoint(int pos);
```

Überschreibt die virtuelle Funktion von *TSlider* und übersetzt eine Positionsangabe in Schieberegler-Einheiten in einen X,Y-Punkt.

Siehe auch *TSlider::PosToPoint*

Klasse TVSlider slider.h

TVSlider, eine von *TSlider* abgeleitete Klasse, stellt Implementationsdetails für vertikale Gleitregler zur Verfügung. Eine Beschreibung eines vertikalen Gleitreglers finden Sie unter *TSlider*.

Öffentliche Konstruktoren

Konstruktor

```
TVSlider(TWindow* parent, int id, int X, int Y, int W, int H, TResId
        thumbResId = IDB_VSLIDERTHUMB, TModule* module = 0);
```

Erzeugt mit einer Standard-Bitmap-Ressourcen-ID von IDB_VSLIDERTHUMB für den Positionsanzeiger-Knopf ein vertikales Schieberegler-Objekt.

Geschützte Elementfunktionen

HitTest

```
int HitTest(TPoint& point);
```

Überschreibt die virtuelle Funktion von *TSlider* und speichert Informationen darüber, wo eine gegebene X,Y-Stelle in den Bereich des Gleitreglers fällt. Die Rückgabe erfolgt als *scrollCodes*-Wert.

Siehe auch *TSlider::HitTest*

NotifyParent

```
void NotifyParent(int scrollCode, int pos = 0);
```

Überschreibt die virtuelle Funktion von *TSlider* und sendet eine WS_VSCOLL-Botschaft an das übergeordnete Fenster.

Siehe auch *TSlider::NotifyParent*

PaintRuler

```
void PaintRuler(TDC& dc);
```

Überschreibt die virtuelle Funktion von *TSlider* und zeichnet das vertikale Lineal.

Siehe auch *TSlider::PaintRuler*

PaintSlot

void PaintSlot(TDC& dc);

Überschreibt die virtuelle Funktion von *TSlider* und zeichnet den Führungsschlitz für den Positionszeiger.

Siehe auch *TSlider::PaintSlot*

PointToPos

```
int PointToPos(TPoint& point);
```

Überschreibt die virtuelle Funktion von *TSlider* und überträgt einen X,Y-Punkt in eine in Slider-Einheiten angegebene Position.

Siehe auch *TSlider::PointToPos*

PosToPoint

```
TPoint PosToPoint(int pos);
```

Überschreibt die virtuelle Funktion von *TSlider* und überträgt eine in Slider-Einheiten angegebene Position in einen X,Y-Punkt.

Siehe auch *TSlider::PosToPoint*

Klasse TSortedStringArray validate.h

TSortedStringArray implementiert eine Liste von ASCII-Zeichen, die als geordnetes Array von Elementen, die String-Objekte sind, gespeichert wird. *TSortedSTringArray* kann viele der String-Manipulationsfunktionen wie beispielsweise das Hinzufügen und Entfernen von Elementen aus dem Array ausführen. Außerdem werden viele der allgemeinen C++ Funktionen, die von Container-Klassen implementiert werden, zur Verfügung gestellt.

Siehe auch Klasse *TValidator*, Klasse *TLookupValidator*

Öffentlicher Konstruktor

Konstruktor

```
TSortedStringArray(int upper, int lower, int delta);
```

Erzeugt ein *TSortedStringArray*-Objekt mit einer oberen Begrenzung *upper*, einer unteren Begrenzung *lower* und einem Wachstumswert *delta*.

Typdefinitionen

void

```
typedef void (*IterFunc)(string&, void*);
```

Funktionstyp, der als Parameter der Elementfunktion *ForEach* verwendet wird.

int

```
typedef int (*CondFunc)(const string&, void*);
```

Funktionstyp, der als Parameter der Elementfunktionen *FirstThat* und *LastThat* verwendet wird.

Öffentliche Elementfunktionen

Add

```
int Add(const string& t);
```

Fügt dem Array an der nächsten verfügbaren Indexposition ein Element hinzu. Das Hinzufügen eines Elements jenseits der oberen Begrenzung führt zu einer Überlauf-Bedingung. Tritt diese Bedingung auf und das vom Wachstumswert *delta* (vom Konstruktor) ist ungleich Null, wird das Array erweitert (um ein ausreichendes Vielfaches von *delta* Bytes), um dem hinzuzufügenden Element Platz zu bieten. Wenn *delta* Null ist, hat *Add* keinen Erfolg. *Add* gibt Null zurück, wenn das Objekt nicht hinzugefügt werden konnte.

ArraySize

```
unsigned ArraySize() const;
```

Gibt die Größe des Arrays zurück.

Destroy

```
int Destroy(const string& t);
```

Entfernt das durch *t* angegebene Element und löscht es.

Destroy

```
int Destroy(int loc);
```

Entfernt ein Element an der angegebenen Indexstelle *loc* aus dem Array und löscht es.

Detach

```
int Detach(const string& t);
```

Entfernt das angegebene Element aus dem Array. Gibt bei Erfolg 1 zurück, andernfalls 0.

Detach

```
int Detach(int loc);
```

Entfernt ein Element an der angegebenen Indexstelle *loc* aus dem Array. Gibt bei Erfolg 1 zurück, andernfalls 0.

Find

```
int Find(const string& t) const;
```

Sucht das erste durch *t* repräsentierte Element und gibt den Index, an dem sich das Element befindet, zurück.

FirstThat

```
string* FirstThat(CondFunc cond, void* args) const;
```

Gibt einen Zeiger auf das erste Element des Arrays zurück, das eine bestimmte Bedingung erfüllt. *cond* ist der Testfunktionszeiger und gibt für eine angegebene Bedingung **true** zurück. *args* enthält die verschiedenen Argumente, die übergeben werden. Gibt 0 zurück, wenn kein Element des Arrays die vorgegebene Bedingung erfüllt. Da *FirstThat* seinen eigenen internen Iterator erzeugt, kann diese Funktion auch als Suchfunktion eingesetzt werden.

Flush

```
void Flush();
```

Erzeugt einen internen Iterator *iter* für jedes Element im Array, der die angegebene Funktion ausführt. Verwenden Sie das Argument *args*, um dieser Funktion verschiedene Arten von Daten zu übergeben.

GetItemsInContainer

```
unsigned GetItemsInContainer() const;
```

Gibt die Anzahl der Elemente im Array zurück.

HasMember

```
int HasMember(const string& t) const;
```

Gibt 1 zurück, wenn das durch *t* angegebene Element im Array vorhanden ist. Andernfalls wird 0 zurückgegeben.

IsEmpty

```
int IsEmpty() const;
```

Gibt 1 zurück, wenn das Array leer ist. Andernfalls wird 0 zurückgegeben.

IsFull

```
int IsFull() const;
```

Gibt 1 zurück, wenn das Array voll ist. Andernfalls wird 0 zurückgegeben.

LastThat

```
string* LastThat(CondFunc cond, void* args) const;
```

Gibt einen Zeiger auf das letzte Element des Arrays zurück, das eine bestimmte Bedingung erfüllt. *cond* ist der Testfunktionszeiger und gibt für eine angegebene Bedingung **true** zurück. *args* enthält die verschiedenen Argumente, die übergeben werden. Gibt 0 zurück, wenn kein Element des Arrays die vorgegebene Bedingung erfüllt. Da *LastThat* seinen eigenen internen Iterator erzeugt, kann diese Funktion auch als Suchfunktion eingesetzt werden.

LowerBound

```
int LowerBound() const;
```

Gibt die untere Begrenzung des Arrays zurück.

operator []

```
string& operator [](int loc);
```

Gibt eine Referenz auf das Element an der durch *loc* angegebenen Stelle zurück. Diese Version paßt - wenn nötig - die Größe des Arrays an, um *loc* zu einem gültigen Index zu machen.

operator []

```
string& operator [](int loc) const;
```

Gibt eine Referenz auf das Element an der durch *loc* angegebenen Stelle zurück. Diese Version löst bei dem Versuch eines Indexeintrages außerhalb der Begrenzungen in der Debugging-Version eine Exception aus.

UpperBound

```
int UpperBound() const;
```

Gibt die obere Begrenzung des Arrays zurück.

Klasse TStatic static.h

TStatic ist ein Oberflächenobjekt, das ein statisches Textoberflächenelement repräsentiert. Statische Elemente bestehen aus Text oder Grafiken, die der Anwender nicht verändert. Eine Anwendung kann jedoch die statischen Dialogelemente verändern. Ein *TStatic*-Objekt muß beispielsweise verwendet werden, um ein statisches Dialogelement in einem übergeordneten *TWindow* zu erstellen, in dem ein Text-Label wie z. B. ein Copyright-Vermerk angezeigt werden soll. Ferner kann es verwendet werden, um Textänderungen in statischen Dialogelementen eines *TDialog*-Objektes zu vereinfachen. Ein Beispiel für statische Dialogelemente finden Sie im Beispielprogramm im Verzeichnis EXAMPLES\OWL\OWLAPI\STATIC.

Öffentliche Datenelemente

TextLen

```
uint TextLen;
```

TextLen speichert die Größe des Textpuffers für statische Dialogelemente. Die tatsächliche Anzahl der Zeichen, die in dem statischen Dialogelement gespeichert werden können, ist um 1 niedriger als *TextLen*, weil der String mit einer 0 abschließt. Ferner gibt *TextLen* die Anzahl der von der Elementfunktion *Transfer* übertragenen Bytes an.

Öffentliche Konstruktoren

Konstruktor

```
TStatic(TWindow* parent, int Id, const char far* title, int x, int y,
        int w, int h, uint textLen, TModule* module = 0);
```

Erstellt ein statisches Dialogelement mit dem angegebenen übergeordneten Fenster (*parent*), der Dialogelement-ID (*Id*), mit dem Text (*title*), der Position (*x*, *y*) im Verhältnis zum Client-Bereich des übergeordneten Fensters, der Breite (*w*), Höhe (*h*) und Textlänge (*textLen*). Per Voreinstellung ist das statische Dialogelement nach der Erstellung sichtbar und sein Text linksbündig ausgerichtet (*Attr.Style* ist auf WS_CHILD | WS_VISIBLE | WS_GROUP | SS_LEFT gesetzt). Die Funktion ruft einen *TControl*-Konstruktor auf. Sie können den Standard-Stil des statischen Dialogelements über den Konstruktor des Dialogelement-Objekts verändern.

Konstruktor

```
TStatic(TWindow* parent, int resourceId, uint textLen, TModule*
        module = 0);
```

Erstellt ein *TStatic*-Objekt, das mit einem statischen Oberflächenelement eines *TDialog*-Objektes verknüpft werden soll. Ruft den *TControl*-Konstruktor mit den entsprechenden Parametern auf und setzt dann *TextLen* auf *textLen*. Setzt durch Aufruf von *DisableTransfer* den Datentransferme-

chanismus außer Kraft. Der *resourceId*-Parameter muß einer von Ihnen zu definierenden Ressource für ein statisches Dialogelement entsprechen.

Siehe auch *TControl::TControl*

Öffentliche Elementfunktionen

Clear

```
virtual void Clear();
```

Löscht den Text des statischen Dialogelements.

GetText

```
int GetText(char far* str, int maxChars);
```

Holt den Text des statischen Dialogelements, speichert ihn im *str*-Argument, das die in *maxChars* angegebene Größe hat, und liefert die Anzahl der kopierten Zeichen zurück.

GetTextLen

```
int GetTextLen();
```

Liefert den Text des statischen Dialogelements zurück.

SetText

```
void SetText(const char far* str);
```

Setzt den Text des statischen Dialogelements auf den in *str* angegebenen String.

Transfer

```
uint Transfer(void* buffer, TTransferDirection direction);
```

Überträgt soviele Zeichen, wie in *TextLen* angegeben ist, in oder aus dem Transferpuffer, auf den *buffer* zeigt. Ist *direction* auf *tdGetData* gesetzt, wird der Text aus dem statischen Dialogelement in den Puffer übertragen. Ist *direction* auf *tdSetData* gesetzt, wird der Text des statischen Dialogelements auf den im Transferpuffer stehenden Text gesetzt. *Transfer* liefert *TextLen* zurück, d.h. die Anzahl der in einen Puffer geschriebenen oder aus einem Puffer geholten Bytes. Wenn *direction* auf *tdSizeData* gesetzt ist, liefert *Transfer TextLen* zurück, ohne Daten zu übertragen.

Geschützte Elementfunktionen

EvSize

```
void EvSize (uint sizeType, TSize& size);
```

Überschreibt die virtuelle Funktion von *TWindow*. Wird die Größe des statischen Dialogelements verändert, stellt *EvSize* sicher, daß es außerdem neu gezeichnet wird.

GetClassName

```
virtual char far* GetClassName();
```

Liefert den Namen der Windows-Registrierungsklasse von *TStatic* (STATIC) oder, wenn BWCC ausgeschaltet ist, STATIC_CLASS zurück.

Antworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_WM_SIZE	EvSize

Klasse TStatus except.h

TStatus wird vorrangig verwendet, um rückwärtsgerichtete Kompatibilität zu älteren Versionen von ObjectWindows herzustellen wird von TModule und *TWindow* verwendet, um einen Fehler bei der Initialisierung eines Oberflächenobjekts anzuzeigen. Ist *Status* auf einen Wert ungleich 0 gesetzt, wird eine *TXCompatibility*-Exception ausgelöst.

Öffentliche Konstruktoren

Konstruktor

```
TStatus();
```

Erstellt ein *TStatus*-Objekt und initialisiert den Statuscode auf 0.

Siehe auch *TModule::Status*, *TWindow::Status*

Öffentliche Datenelemente

operator=

```
inline TStatus& operator=(int statusCode);
```

Setzt den Statuscode und löst eine *TXCompatibility*-Exception aus.

operator

```
inline operator int() const;
```

Liefert den Statuscode zurück.

Klasse TStatusBar statusba.h

TStatusBar bietet Unterstützung für Statuszeilen, eingefügte Gadgets, Hinweistext und Tastaturmodi. Im Gegensatz zu reinen Meldungszeilen stellen Statuszeilen mehrere Anzeigeoptionen zur Verfügung. ObjectWindows-Statuszeilen erlauben Ihnen die Einbeziehung zahlreicher Text-Gadgets (der Text auf der linken Seite der Statuszeile) und verschiedener Rahmenstile. Außerdem können Sie Platz für Modusanzeigen (der Text, der den aktuellen Programmzustand anzeigt) reservieren, wie etwa erweiterte Auswahl (von Tasten und anderen Modi), *CapsLock*, *NumLock*, *ScrollLock*, *Überschreiben* und Aufzeichnen von Makros. ObjectWindows

verwendet diese virtuellen Tastaturcodes per Voreinstellung und zeigt für die angegebenen aktivierten Modi die folgenden Strings an:

Modus	Virtueller Tastaturcode	Angezeigter Text
Erweitert		EXT
CapsLock	VK_CAPITAL	CAPS
NumLock	VK_NUMLOCK	NUM
ScrollLock	VK_SCROLL	SCRL
Überschreiben	VK_INSERT	OVR
Makro aufzeichnen		REC

TStatusBar erzeugt Text-Gadgets für die von Ihnen geforderten Modusanzeigen und richtet den Raum zwischen Modusanzeigen ein. Die Struktur *TSpacing* speichert Randbedingungen für Abstands- und Layouteinheiten.

Wie andere Dialogelementleisten auch, wird die Statuszeile zu derselben Zeit wie das übergeordnete Fenster erstellt und gelöscht, doch ist dies keine erforderliche Prozedur.

Die folgenden Programmanweisungen zeigen, wie eine Statuszeile erstellt und unten in das Fenster eingefügt wird.

```
TStatusBar* sb = new TStatusBar(0, TGadget_Recessed,
        TStatusBar_CapsLock | TStatusBar_NumLock |
        TStatusBar_Overtype);
frame->Insert(*sb, TDecoratedFrame::Bottom);
MainWindow = frame;
```

Ein Beispielprogramm (MDIFILE.CPP) für das Erstellen eines Fensters mit einer Statuszeile finden Sie auf Ihrer Beispieldiskette.

Typdefinition

TModeIndicator

```
enum TModeIndicator {ExtendSelection = 1, CapsLock = 1 << 1,
                     NumLock = 1 << 2, ScrollLock = 1 << 3,
                     Overtype = 1 << 4, RecordingMacro = 1 << 5};
```

Zählt die Tastaturmodi auf. Per Voreinstellung sind diese horizontal von links nach rechts auf der Statuszeile angeordnet. Setzt die erweiterte Auswahl *CapsLock, NumLock, ScrollLock, Überschreiben* und die Anzeigen zum Aufzeichnen von Makros.

Öffentliche Konstruktoren

Konstruktor

```
TStatusBar(TWindow* parent = 0,
           TGadget_TBorderStyle borderStyle = TGadget_Recessed,
           uint modeIndicators = 0, TFont *font = new TGadgetWindowFont,
           TModule* module = 0);
```

Erstellt ein *TStatusBar*-Objekt im übergeordneten Fenster und erzeugt jede Art von neuen Gadgets und Modusanzeigen-Gadgets. Setzt *BorderStyle* auf *borderStyle*, *ModeIndicators* auf *modeIndicators* und *NumModeIndicators* auf 0. borderStyle kann jeder Wert des Aufzählungstyps *TBorderStyle* sein (normal, hervorgehoben, zurückgesetzt oder geprägt). Die Anzeigen der Parameter-Modi können einer der Werte des Aufzählungstyps *TModeIndicator* sein wie beispielsweise CapsLock, NumLock, ScrollLock oder Überschreiben. Der Parameter *font* zeigt auf ein Font-Objekt, das die im Gadget-Fenster verwendeten Schriftarten enthält. Der Parameter, *module*, standardmäßig 0, wird im Parameter *module* an den Basiskonstruktor von *TWindow* übergeben. Setzt die Werte für Ränder und Rahmen abhängig davon, ob das Gadget erhaben, vertieft oder normal ist.

Öffentliche Elementfunktionen

GetModeIndicator

```
bool GetModeIndicator(TModeIndicator i) const;
```

Gibt den aktuellen Modus der Statuszeile zurück.

Insert

```
void Insert(TGadget& gadget, TPlacement = After, TGadget* sibling = 0);
```

Fügt das Gadget (von der Klasse *TGadget* abgeleitete Objekte) in die Statuszeile ein. Per Voreinstellung wird jedes neue Gadget direkt hinter bereits existierende Gadgets und zur Linken der Modusanzeigen in der Statuszeile plaziert. Beispielsweise können Sie ein Zeichenwerkzeug oder ein Symbol, das ein aufgezeichnetes Makro aktiviert, einfügen.

operator[]

```
TTextGadget* operator[](uint index);
```

Liefert ein Gadget zu einem gegebenen Index zurück, hat jedoch keinen Zugriff auf Modusanzeige-Gadgets.

SetModeIndicator

```
void SetModeIndicator(TModeIndicator, bool state);
```

Setzt *TModeIndicator* auf ein gegebenes Text-Gadget und den Status (voreingestellt ist "an") auf die Modusanzeige. Damit die Modusanzeige in der Statuszeile erscheint, müssen Sie den Modus beim Erstellen des Fensters festlegen.

Siehe auch *TStatusBar::TModeIndicator*

SetSpacing

```
void SetSpacing(TSpacing& spacing);
```

Verwendet die Werte in *TSpacing*, um die Abstände zwischen Modusanzeige-Gadgets einzurichten. *TSpacing* setzt die Ränder der Statuszeile in Layouteinheiten. Typischerweise ist die Meldungsanzeige (das am weitesten links liegende Text-Gadget) links und die anderen Anzeigen rechts in der Statuszeile angeordnet. Eine detaillierte Erläuterung zu Layouteinheiten und Randbedingungen finden Sie unter *TLayoutMetrics*.

```
struct TSpacing {
  TMargins::TUnits Units;
  int Value;
  TSpacing() {Units = TMargins::LayoutUnits; Value = 0;}
};
```

Siehe auch *TStatusBar::TModeIndicator*

ToggleModeIndicator

```
void ToggleModeIndicator(TModeIndicator);
```

Schaltet *ModeIndicator* um.

Geschützte Datenelemente

BorderStyle

```
TGadget::TBorderStyle BorderStyle;
```

Eine der aufgelisteten Rahmenarten - keiner, flach, erhaben, vertieft oder geprägt -, die von den Modusanzeigen in der Statuszeile verwendet werden.

ModeIndicators

```
uint ModeIndicators;
```

Das *ModeIndicators*-Bitfeld zeigt an, welche Modusanzeigen für die Statuszeile erstellt wurden.

ModeIndicatorState

```
uint ModeIndicatorState;
```

Gibt den Modus der Statuszeile an. Dies kann jeder der Werte des Aufzählungstypes *TModeIndicator* wie beispielsweise CapsLock, ScrollLock, Überschreiben, *RecordingMacro* oder *ExtendSelection* sein.

NumModeIndicators

```
uint NumModeIndicators;
```

Bestimmt die Anzahl der Modusanzeigen. Diese kann von 1 bis 5 reichen.

Spacing

```
TSpacing Spacing;
```

Bestimmt die Größe der Abstände zwischen Modusanzeigen in der Statuszeile.

Geschützte Elementfunktionen

IdleAction

```
bool IdleAction(long);
```

Wenn mehr als eine Anwendung läuft, ruft *TStatusBar* die Funktion *IdleAction* an Stelle von *PreProcessMsg* auf, um den Zustand der Tasten *NumLock*, *CapsLock* oder *ScrollLock* zu testen und um die Modusindikatoren zu aktualisieren, falls sie nicht den aktuellen Zustand dieser Tasten reflektieren.

PositionGadget

```
void PositionGadget(TGadget* previous, TGadget* next, TPoint& point);
```

Ermittelt die Position des neuen Gadget im Verhältnis zu möglichen bereits existierenden Gadgets und verwendet die *Pixels-*, *LayoutUnits-*, und

BorderUnits-Felder von *TMargins*, um die Größe der Abstände zu bestimmen, die zwischen den Modusanzeigen gelassen werden sollen.

PreProcessMsg

```
bool PreProcessMsg(MSG& msg);
```

Überschreibt *TWindow::PreProcessMsg*, um Tastaturbotschaften zu verarbeiten und um die Modusindikatoren zu aktualisieren, wenn die Tasten *NumLock*, *CapsLock* oder *ScrollLock* gedrückt wurden.

Klasse TStorageDocument stgdoc.h

TStorageDocument ist von *TDocument* abgeleitet und stellt Funktionen zur Verfügung, die die Verbunddatei-Struktur von OLE unterstützen. Eine Verbunddatei-Struktur ist ein Dateiverwaltungssystem, das Dateien in einer hierarchischen Struktur innnerhalb einer Stammdatei speichert. Diese Speicherstruktur ist analog der Speicherstruktur, die auf einer Festplatte verwendet wird und mit Verzeichnissen oder Ordnern und Dokumenten arbeitet. Der Unterschied besteht nur darin, daß man ein Verzeichnis einen Speicherplatz nennt und eine Datei einen Stream.

Außerem bietet *TStorageDocument* Unterstützung für den Verbunddokument-Mechanismus von OLE. Ein Verbunddokument kann viele unterschiedliche eingebettete Objekte speichern, beispielsweise Arbeitsblätter und Bitmap-Grafiken.

Grundsätzlich unterstützt *TDocument* das Lesen und Schreiben des eigenen Speicherplatzes durch das Dokument. Um diese Funktionen anbieten zu können, überschreibt *TStorageDocument* eine Reihe von virtuellen Methoden aus *TDocument*. Außerdem werden gemäß der Strategie von *TDocument* Eigenschaftslisten sowohl für Dokumente als auch für ihre Views verwendet. Auf diese Weise können Dokumente die Attribute dazu benutzen, Dateien aus dem Speicherplatz einzulesen oder in den Speicherplatz zu schreiben.

Es werden Botschaften an die Anwendung gesandt, die die Eigenschaften abfragt, um die Verarbeitung des Dokuments oder der View festzulegen. Jede abgeleitete Klasse muß ihre eigenen Eigenschafts-Attributtypen implementieren - entweder als Strings oder als Binärdaten.

Siehe auch *TDocument, TOleDocument*

Typdefinitionen

TStgDocProp

```
enum TStgDocProp {PrevProperty, CreateTime, ModifyTime, AccessTime,
StorageSize, IStorageInstance, NextProperty};
```

Zählt die folgenden Konstanten auf, die die Eigenschaften des Dokuments definieren:

Bedeutung

PrevPropertySollte immer den Wert von *TDocument::NextProperty* -1 besitzen. Dies ist der Initialwert der Eigenschaftsliste jedes Dokuments.
CreateTime Zeitpunkt, zu dem die Datei erstellt wurde.
ModifyTime Zeitpunkt, zu dem die Datei geändert wurde.
AccessTime Zeitpunkt, zu dem zuletzt auf die Datei zugegriffen wurde.
IStorageInstance Speicherinstanz für das Dokument.
NextProperty Der letzte Wert der Eigenschaftsliste jedes Dokuments.

Öffentlicher Konstruktor und Destruktor

Konstruktor

```
Constructor TStorageDocument(TDocument* parent = 0):TDocument(parent),
StorageI(0), OpenCount(0), CanRelease(false);
```

Erzeugt ein *TStorageDocument*-Objekt mit dem angegebenen übergeordneten *TDocument*-Fenster. Setzt die Datenelemente *StorageI* und *OpenCount* auf 0. Setzt *CanRelease* auf **false**. Später setzt die Elementfunktion *ReleaseDoc* dieses Flag auf **true**, damit daß Dokument geschlossen werden kann.

Destruktor

```
~TStorageDocument();
```

Entfernt ein *TStorageDocument*-Objekt.

Öffentliche Elementfunktionen

Close

```
bool Close();
```

Gibt *IStorage* frei, wenn *CanRelease* **true** ist. (*CanRelease* wird auf **true** gesetzt, wenn *ReleaseDoc* aufgerufen wird.) Bevor das Dokument geschlossen wird, überprüft *Close* alle untergeordneten Dokumente und versucht, diese zu schließen.

Siehe auch *TStorageDocument::Open, TStorageDocument::ReleaseDoc, TOutStream, ofxxxx-Dokument-Öffnen*-Aufzählungstyp

Commit

```
bool Commit(bool force = false);
```

Speichert die aktuellen Daten im Speicherbateil. Wird eine Datei geschlossen, ruft der Dokumentverwalter entweder *Commit* oder *Revert* auf. Wenn *force* **true** ist, werden alle Daten gespeichert. *TDocument*s *Commit* überprüft alle untergeordneten Dokumente und veranlaßt auch ihre Speicherung. Bevor die aktuellen Daten gespeichert werden, müssen alle untergeordneten Dokumente **true** zurückgeben. Wenn alle untergeordneten Dokumente **true** zurückgeben, leert *Commit* die Views von allen Operationen, die seit der letzten Überprüfung der View aufgetreten sind. Sobald alle Daten für das Dokumentobjekt aktualisiert und gespeichert wurden, gibt *Commit* **true** zurück.

FindProperty

```
int FindProperty(const char far* name);
```

Holt bei gegebenem Eigenschaftsnamen den Eigenschaftsindex. Gibt die Integer-Indexnummer, die dem Namen entspricht, zurück. Wird der Name in der Eigenschaftsliste nicht gefunden, wird 0 zurückgegeben.

Siehe auch *pfxxxx-Eigenschaftsattribut*-Konstanten

GetProperty

```
int GetProperty(int index, void far* dest, int textlen=0);
```

Gibt die Gesamtzahl der Eigenschaften für dieses Speicherplatz-Dokument zurück, wobei *index* der Eigenschaftsindex ist, *dest* die Eigenschaftsdaten enthält und *textlen* die Größe des Eigenschafts-Arrays ist. Wenn *textlen* 0 ist, werden die Eigenschaftsdaten als Binärdaten zurückgeliefert. Anderfalls werden die Eigenschaftsdaten als Textdaten zurückgeliefert.

Siehe auch *pfxxxx-Eigenschaftsattribut*-Konstanten, *TStorageDocument::SetProperty*

GetStorage

```
IStorage* GetStorage();
```

Gibt den Stamm-*IStorage* des Dokuments zurück.

InStream

```
TInStream* InStream(int omode, const char far* strmId=0);
```

Stellt einen Eingabe-Stream für das Objekt *TStorageDocument* zur Verfügung. Gibt einen Zeiger auf ein *TInStream*-Objekt zurück. Der Parameter *omode* enthält eine Kombination der Modi zum Dokument Öffnen und für gemeinsamen Dateizugriff (beispielsweise *ofReadWrite*), die in docview.h definiert sind. Der Parameter *strmId* wird für Dokumente verwendet, die benannte Streams unterstützen.

Siehe auch *TStorageDocument::OutStream, TInStream, ofxxxx-Dokument-Öffnen*-Aufzählungstyp

IsOpen

```
bool IsOpen();
```

Überprüft, ob das Speicherplatz-Dokument irgendein *IStorage* erzeugt hat. Gibt es keinen Speicherplatz, gibt *IsOpen* **false** zurück. Andernfalls wird **true** zurückgegeben.

Open

```
bool Open(int omode, const char far* name);
```

Öffnet oder erzeugt ein Dokument, das auf *IStorage* basiert. Der Parameter *name* gibt den Namen des Dokuments - wenn vorhanden - an, das geöffnet werden soll. Der Parameter *omode* enthält eine Kombination der Modi zum Dokument Öffnen und für gemeinsamen Dateizugriff (beispielsweise *ofReadWrite*), die in docview.h definiert sind.

Siehe auch *TStorageDocument::Close, TOutStream, ofxxxx-Dokument-Öffnen*-Aufzählungstyp

OpenHandle

```
virtual bool OpenHandle(int omode, HANDLE hGlobal);
```

OpenHandle schreibt Daten in einen Speicherblock. *OpenHandle* erzeugt zuerst ein *ILockBytes*-Oberflächenobjekt (eine OLE2-Schnittstelle, die Lesen, Schreiben und Sperren einer Folge von Bytes in einer Datei implementiert) auf dem globalen Handle und anschließend ein *IStorage*, das auf dem *ILockBytes*-Oberflächenelement basiert. Der Parameter *omode* enthält eine Kombination der Modi zum Dokument Öffnen und für gemeinsamen Dateizugriff (beispielsweise *ofReadWrite*), die in docview.h definiert sind.

Siehe auch *ofxxxx-Dokument-Öffnen*-Aufzählungstyp

OutStream

```
TOutStream* OutStream(int omode, const char far* strmId=0);
```

Stellt einen Ausgabe-Stream für ein bestimmtes Speichermedium zur Verfügung. Gibt einen Zeiger auf einen *TOutStream* zurück. Der Parameter *omode* enthält eine Kombination der Modi zum Dokument Öffnen und für gemeinsamen Dateizugriff (beispielsweise *ofReadWrite*), die in docview.h definiert sind. Der Parameter *strmId* wird für Dokumente verwendet, die benannte Streams unterstützen.

Siehe auch *TStorageDocument::InStream, TOutStream, ofxxxx-Dokument-Öffnen*-Aufzählungstyp

PropertyCount

```
int PropertyCount();
```

Holt die Gesamtzahl der Eigenschaften für das *TStorageDocument*-Objekt. Gibt *NextProperty* -1 zurück.

Siehe auch *pfxxxx-Eigenschaftsattribut*-Konstanten

PropertyFlags

```
int PropertyFlags(int index);
```

Gibt die Attribute einer angegebenen Eigenschaft bei gegebenem Index der Eigenschaft, deren Attribute ermittelt werden sollen, an.

Siehe auch *pfxxxx-Eigenschaftsattribut*-Konstanten

PropertyName

```
const char* PropertyName(int index);
```

Gibt den Namen der Eigenschaft bei gegebenem Indexwert zurück.

Siehe auch *pfxxxx-Eigenschaftsattribut*-Konstanten

ReleaseDoc

```
virtual bool ReleaseDoc();
```

Gibt den Speicherplatz für das Dokument frei und schließt das Dokument.

Revert

```
bool Revert(bool clear = false),
```

Revert ist das Gegenstück zu *Commit*. *Revert* hebt alle Änderungen, die an dem Speicherplatz-Dokument seit dem letzten *Commit* vorgenommen wurden, auf. Wenn *clear* **true** ist, werden Daten für Views nicht neu geladen. *Revert* prüft außerdem alle untergeordneten Dokumente und hebt alle Änderungen auf, wenn alle untergeordneten Dokumente **true** zurück-

geben. Wird eine Datei geschlossen, ruft der Dokumentverwalter entweder *Commit* oder *Revert* auf. *Revert* gibt **true** zurück, wenn die *Revert*-Operation erfogreich war.

SetDocPath

```
bool SetDocPath(const char far* path);
```

Setzt den Dokumentpfad für die Dateioperationen Öffnen und Speichern.

SetProperty

```
bool SetProperty(int index, const void far* src);
```

Setzt den Wert der Eigenschaft bei gegebenem Index der Eigenschaft und *src*, dem Datentyp (entweder binär oder Text), auf den die Eigenschaft gesetzt werden muß.

Siehe auch *pfxxxx-Eigenschaftsattribut*-Konstanten, *TStorageDocument::GetProperty*

SetStorage

```
IStorage* SetStorage(IStorage* stg);
```

Versieht dieses Dokument mit einem *IStorage*-Zeiger (*stg*).

Siehe auch *TStorageDocument::GetStorage, TStorageDocument::StorageI*

Geschützte Datenelemente

ThisOpen

```
int ThisOpen;
```

Enthält die tatsächlichen Modus-Bits, die zum Öffnen des Speicherplatzes verwendet werden. Die Modus-Bits bestimmen, wie eine Datei geöffnet wird: beispielsweise nur Lesezugriff, Lese- und Schreibzugriff usw.

Siehe auch *ofxxxx-Dokument-Öffnen*-Aufzählungstyp

StorageI

```
IStorage* StorageI;
```

Enthält die aktuelle Instanz von *IStorage*. (Ist kein Speicherplatz geöffnet, ist *StorageI* 0.)

Siehe auch *TStorageDocument::GetStorage, TStorageDocument::SetStorage*

Klasse TStream — docview.h

TStream, eine abstrakte Klasse, stellt Verknüpfungen zwischen Streams und Dokumenten, View-Objekten und Dokumentdateien her.

Öffentlicher Destruktor

Destruktor

```
inline ~TStream();
```

Schließt den Stream. Abgeleitete Klassen schließen normalerweise das Dokument, wenn es speziell für diesen Stream geöffnet wurde.

Öffentliche Elementfunktionen

GetDocument

```
TDocument& GetDocument();
```

Liefert das für das Streaming geöffnete aktuelle Dokument zurück.

GetOpenMode

```
int GetOpenMode();
```

Ermittelt die Modus-Flags, die beim Öffnen von Dokumenten-Streams verwendet werden. Der Stream kann zum Beispiel im Modus *ofRead* geöffnet werden, wodurch ein Lesezugriff auf die Datei erlaubt ist. Ein Ändern der Datei (Schreibzugriff) dann ist jedoch nicht möglich.

GetStreamName

```
const char far* GetStreamName();
```

Ermittelt den Namen des Streams, der beim Öffnen des Dokumentes verwendet wird.

Geschützte Datenelemente

Doc

```
TDocument& Doc;
```

Speichert das Dokument, das diesen Stream besitzt.

NextStream

```
TStream* NextStream;
```

Zeigt auf den nächsten Stream in der Liste der aktiven Streams.

Geschützte Konstruktoren

Konstruktor

```
TStream(TDocument& doc, const char far* name,int mode);
```

Erstellt in *TStream*-Objekt. *doc* referiert auf das Dokumentobjekt, *name* ist der benutzerdefinierte Name des Streams und *mode* der Modus, der zum Öffnen des Streams verwendet wurde.

Siehe auch *TInStream*, *TOutStream*, *ofXXXX document open*-Aufzählungstyp, *shXXXX document sharing*-Aufzählungstyp

Klasse TStringLookupValidator — validate.h

TStringLookupValidator ist eine von *TLookupValidator* abgeleitete streamfähige Klasse. Ein *TStringLookupValidator*-Objekt überprüft die Daten in dem mit ihm verknüpften Editierfeld, indem es eine Auswahl gültiger Strings durchsucht. Sie können Validatoren zur Stringsuche verwenden, wenn Ihr Editierfeld nur Strings aus einer bestimmten Stringmenge akzeptieren soll.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TStringLookupValidator(TSortedStringArray strings);
```

Konstruiert ein Objekt zur Stringsuche, indem zunächst der von *TLookupValidator* geerbte Konstruktor aufgerufen und dann *Strings* auf *strings* gesetzt wird.

Destruktor

```
~TStringLookupValidator();
```

Konstruiert eine Liste gültiger Strings durch Aufruf von *NewStringList* und löscht dann das Validator-Objekt zur Stringsuche, indem es den von *TLookupValidator* geerbten Destruktor aufruft.

Öffentliche Elementfunktionen

Error

```
void Error();
```

Überschreibt die virtuelle Funktion von *TValidator* und öffnet ein Meldungsfenster mit der Nachricht, daß der eingegebene String keinem Eintrag in der Liste der Strings entspricht.

Siehe auch *TValidator*::Error

Lookup

```
bool Lookup(const char far* str);
```

Überschreibt die virtuelle Funktion von *TLookupValidator*. Liefert **true** zurück, wenn der in *str* übergebene String einem der *strings* entspricht. Verwendet die Suchmethode der jeweiligen Stringsammlung, um festzustellen, ob *str* vorhanden ist.

Siehe auch *TLookupValidator::Lookup*

NewStringList

```
void NewStringList(TSortedtringArray strings);
```

Setzt die Liste gültiger Eingabestrings für den Stringsuche-Validator. Entfernt möglicherweise bereits vorhandene Stringlisten und setzt dann *Strings* auf *strings*.

Geschütztes Datenelement

Strings

```
TSortedStringArray Strings;
```

Zeigt auf eine Stringsammlung, die alle gültigen Strings enthält, die der Benutzer eingeben kann. Ist *Strings* NULL, wird jede Eingabe geprüft.

Klasse TSystemMenu — menu.h

TSystemMenu erstellt ein Systemmenü-Objekt, das darauf zum bestehenden Systemmenü wird.

Öffentliche Konstruktoren

Konstruktor

```
TSystemMenu(HWND wnd, bool revert = false);
```

Erstellt ein Systemmenü-Objekt. Ist *revert* **true**, so ist das erzeugte Menü ein voreingestelltes Systemmenü; andernfalls ist es das gegenwärtig im Fenster befindliche.

Siehe auch *TPopupMenu::TPopupMenu*

Klasse TTextGadget textgadg.h

TTextGadget ist ein aus *TGadget* abgeleitetes Text-Gadget-Objekt. Wenn Sie ein neues Text-Gadget erstellen, müssen Sie festlegen, für wieviele Zeichen Platz reserviert werden und in welcher Weise der Text horizontal ausgerichtet sein soll. Die inneren Begrenzungen des Text-Gadget werden durch Multiplikation der Anzahl von Zeichen mit der maximalen Zeichenbreite errechnet.

Öffentliche Datenelemente

TAlign

```
enum TAlign {Left, Center, Right};
```

Listet die Textausrichtungsattribute auf. *Left* richtet den Text an der linken Kante des begrenzenden Rechtecks aus, *Right* an der rechten. *Center* zentriert den Text horizontal in dem begrenzenden Rechteck.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TTextGadget(int id = 0, TBorderStyle = Recessed, TAlign = Left, uint
            numChars = 10, const char* text = 0);
```

Erzeugt ein *TTextGadget*-Objekt mit festgelegter ID, Rahmenart und Ausrichtung. Setzt *Margins.Left* und *Margins.Right* auf 2. Setzt *Text* und *TextLen* auf 0.

Destruktor

```
~TTextGadget();
```

Entfernt ein *TTextGadget*-Objekt.

Öffentliche Elementfunktionen

GetText

```
inline char* GetText();
```

Liefert den Text für das Gadget zurück.

SetText

```
void SetText(const char* text);
```

Wenn der in *Text* gespeicherte Text nicht der gleiche wie der neue Text ist, löscht *SetText* den in *Text* gespeicherten Text und setzt darauf *TextLen* auf die Länge des neuen Strings. Liegt kein Text vor, werden *Text* und *TextLen* auf 0 gesetzt und ein Aufruf von *Invalidate* erfolgt, um das Rechteck ungültig zu machen.

Geschützte Datenelemente

Align

```
TAlign Align;
```

Textausrichtungsattribut - linksbündige, zentrierte oder rechtsbündige Ausrichtung.

NumChars

```
uint NumChars;
```

Speichert die Anzahl der Textzeichen.

Text

```
char* Text;
```

Zeigt auf den Text für das Gadget.

TextLen

```
uint TextLen;
```

Speichert die Länge des Texts.

Geschützte Elementfunktionen

GetDesiredSize

```
void GetDesiredSize(TSize &size);
```

Wird eine Anpassung des Fensters gewünscht, liefert *GetDesiredSize* die für die Einstellung von Rahmen, Rändern und Text benötigten Maße zurück; andernfalls, d.h., wenn keine Anpassung gewünscht wird, liefert die Funktion die aktuelle Breite und Höhe des Gadget zurück.

Siehe auch *TGadget::GetDesiredSize*

Invalidate

```
void Invalidate();
```

Ruft *TGadget::GetInnerRect* auf, um den Textbereich für das Gadget zu errechnen und daraufhin *TGadget::InvalidateRect*, um das Rechteck im übergeordneten Fenster zu löschen.

Siehe auch *TGadget::GetInnerRect*, *TGadget::Invalidate*

Paint

```
void Paint(TDC& dc);
```

Ruft *TGadget::PaintBorder* auf, um den Rahmen zu zeichnen. Ruft *TGadget::GetInnerRect* auf, um den Bereich des Rechtecks für das Text-Gadget zu errechnen. Ist der Text linksbündig ausgerichtet, ruft *Paint dc.GetTextExtent* auf, um die Breite und Höhe einer Textzeile zu berechnen. Um die Hintergrundfarbe zu setzen, ruft *Paint dc.GetSysColor* auf und setzt die Standard-Hintergrundfarbe auf Schrifttönung (COLOR_BTNFACE). Um die Farbe für den Text des Schaltfeldes zu setzen,

ruft *Paint dc.SetTextColor* auf und setzt die Standard-Schaltfeldtextfarbe auf COLOR_BTNTEXT. Um den Text auszugeben, ruft *Paint dc.ExtTextOut* auf und übergibt die Parameter ETO_CLIPPED (damit wird der Text so beschnitten, daß er in das Rechteck paßt) und ETO_OPAQUE (damit wird das Rechteck mit der aktuellen Hintergrundfarbe gefüllt).

Siehe auch *TGadget::Paint*

Typdefinition TTileDirection gadgetwi.h

TTileDirection

```
enum TTileDirection{Horizontal, Vertical};
```

Listet die horizontale und vertikale Richtung für die Anordnung des Gadget neben den anderen auf. *TGadgetWindow::TileGadgets* ordnet die Gadgets in der gewünschten Richtung nebeneinander an.

Siehe auch *TGadgetWindow::Direction, TGadgetWindow::TileGadgets*

TAlign

```
enum TAlign{Left, Center, Right};
```

Listet die Attribute der Textausrichtung auf. *Left* richtet den Text an der linken Kante des begrenzenden Rechtecks aus. *Right* richtet den Text an der rechten Kante des begrenzenden Rechtecks aus. *Center* richtet den Text horizontal an der Mitte des begrenzenden Rechtecks aus.

Klasse TTinyCaption tinycapt.h

TTinyCaption ist eine von *TWindow* abgeleitete Mix-in-Klasse, die eine Reihe von Nicht-Client-Ereignissen verarbeitet, um eine kleinere Titelleiste für ein Fenster zu erstellen. Immer wenn *TTinyCaption* die Titelleiste anzeigt, überprüft es den Fensterstil und bearbeitet die Windows-Anzeigenattribute

WS_SYSMENU, WS_MINIMIZEBOX und WS_MAXIMIZEBOX. Somit können Sie *TTinyCaption* verwenden, um die Attribute der kleinen Titelleiste zu setzen, bevor der Titel aktiviert wird. Zum Beispiel:

```
Attr.Style = WS_POPUP | WS_BORDER | WS_SYSMENU | WS_MINIMIZEBOX |
             WS_MAXIMIZEBOX;
```

TTinyCaption stellt Funktionen zur Verfügung, mit denen Sie Windows-Rahmenarten und -stile sowie Menüs bearbeiten können. Sie können die Höhe der Titelleiste einstellen oder die voreingestellte Höhe übernehmen, die etwa der halben Höhe einer Standard-Titelleiste entspricht. Wenn Sie *CloseBox* auf **true** setzen, schließt sich das Fenster, anstatt das Systemmenü anzuzeigen, wenn Sie das Schließsymbol anklicken.

Die Datei OWLCMD.CPP befindet sich auf Ihrer Beispieldiskette.

Wenn Sie *TTinyCaption* als Mix-in-Klasse für partielles Ereignisbehandlung verwenden, rufen Sie (anstelle der *EvXxxx*-Funktion) die *DoXxxx*-Funktion der Mix-in-Klasse auf, um eine Verdopplung der Vorgabeberechnungen zu umgehen. Der folgende Ausschnitt aus OWLCMD.CPP (einem Beispielprogramm auf Ihrer Beispieldiskette) illustriert diesen Vorgang:

```
void TMyFrame::EvSysCommand(uint cmdType,TPoint& p)
{
if (TTinyCaption::DoSysCommand(cmdType, p) == esPartial)
    TFrameWindow::EvSysCommand(cmdType, p);
};
```

Die Klasse *TFLoatingFrame* kann zusammen mit *TTinyCaption* verwendet werden, um ein Schließsymbol zu erstellen. Die Beispielprogramme OWLCMD.CPP und MDIFILE.CPP erläutern die Verwendung von *TTinyCaption*.

Geschützte Datenelemente

Border

```
TSize Border;
```

Schmale Rahmenart für Trennlinien.

CaptionFont

```
TFont* CaptionFont;
```

Für den Text der kleinen Titelleiste verwendete Schriftart.

CaptionHeight

```
int CaptionHeight;
```

Höhe der Titelleiste

CloseBox

```
bool CloseBox;
```

Ist *CloseBox* **true**, schließt sich das Fenster, wenn das Schließsymbol angeklickt wird.

DownHit

```
uint DownHit;
```

Stelle, an der eine Maustaste gedrückt oder der Cursor bewegt wurde.

Frame

```
TSize Frame;
```

Tatsächliche Ausdehnung der Titelleiste nach links und rechts, oben und unten.

IsPressed

```
bool IsPressed;
```

Ist **true**, wenn eine Maustaste gedrückt wird.

TCEnabled

```
bool TCEnabled;
```

Ist **true**, wenn die kleine Titelleiste angezeigt wird.

WaitingForSysCmd bool WaitingForSysCmd;

Ist **true**, wenn *TTinyCaption* zum Empfang von Systembotschaften bereit ist.

Geschützte Konstruktoren und Destruktor

Konstruktor

```
TTinyCaption();
```

Erzeugt ein *TTinyCaption*-Objekt, das mit dem gegebenen übergeordneten Fenster verbunden ist. Initialisiert die Titelleisten-Schriftart mit 0 und setzt *TCEnabled* auf **false**, so daß die kleine Titelleiste nicht automatisch angezeigt wird.

Destruktor

```
~TTinyCaption();
```

Entfernt ein *TTinyCaption*-Objekt und löscht die Titelleisten-Schriftart.

Geschützte Elementfunktionen

DoCommand

```
TEventStatus DoCommand(uint id, HWND hWndCtl, uint notifyCode, LRESULT&
evRes);
```

Zeigt das Systemmenü unter Verwendung von *::TrackPopup* an, so daß *TTinyCaption* WM_COMMAND- an Stelle von WM_SYSCOMMAND-Botschaften sendet. Wird ein Befehl des Systemmenüs empfangen, wird er somit in eine WM_SYSCOMMAND-Botschaft umgewandelt. Ist der Wert der kleinen Titelleiste **false**, liefert *DoCommand esPartial* zurück.

Siehe auch *TTinyCaption::EvCommand*, *TEventStatus*-Aufzählungstyp

DoLButtonUp

```
TEventStatus DoLButtonUp(uint hitTest, TPoint& screenPt);
```

Löst die Blockierung der Maus, wenn die Titelleiste aktiviert und eine Maustaste gedrückt ist. Setzt *hitTest*, womit das Drücken der Maustaste angezeigt wird. Fängt die Mausbotschaft ab und zeichnet die kleineren Schalter neu, bevor *esComplete* zurückgegeben wird.

Siehe auch *TTinyCaption::EvLButtonUp*

DoMouseMove

```
TEventStatus DoMouseMove(uint hitTest, TPoint& screenPt);
```

Liefert *TEventStatus* zurück.

DoNCActivate

```
TEventStatus DoNCActivate(bool active, bool& evRes);
```

Wenn die kleine Titelleiste deaktiviert oder als Symbol dargestellt ist, wird *esPartial* zurückgeliefert. Andernfalls zeichnet *DoNCActivate* die Titelleiste neu als aktive Titelleiste und liefert *esComplete* zurück.

Siehe auch *TTinyCaption::EvNCActivate*

DoNCCalcSize

TEventStatus DoNCCalcSize(bool calcValidRects, NCCALCSIZE_PARAMS far&calcSize, uint& evRes);

Wenn die Titelleiste deaktiviert oder als Symbol dargestellt ist, wird *esPartial* zurückgeliefert. Andernfalls berechnet *DoNCActivate* die Ausmaße der Titelleiste und liefert *esComplete* zurück.

Siehe auch *TTinyCaption::EvNCCalcSize*

DoNCHitTest

```
TEventStatus DoNCHitTest(TPoint& screenPt, uint& evRes);
```

Ist die Titelleiste nicht aktiviert, wird *esPartial* zurückgeliefert. Andernfalls sendet *DoNCHitTest* eine Botschaft an die Titelleiste, daß die Maus oder der Cursor bewegt wurde und liefert *esComplete* zurück.

Siehe auch *TTinyCaption::EvNCHitTest*

DoNCLButtonDown

```
TEventStatus DoNCLButtonDown(uint hitTest, TPoint& screenPt);
```

Ist die Titelleiste nicht aktiviert, wird *esPartial* zurückgeliefert. Andernfalls ermittelt *DoLButtonDown*, ob der Benutzer die Taste außerhalb oder innerhalb eines Menüs losgelassen hat und liefert *esComplete* zurück.

Siehe auch *TTinyCaption::EvNCLButtonDown*

DoNCPaint

```
TEventStatus DoNCPaint();
```

Liefert *esPartial* zurück, wenn die Titelleiste nicht aktiviert oder als Symbol dargestellt ist. Andernfalls erhält es den Fokus, die Titelleiste wird gezeichnet und *esPartial* zurückgeliefert, womit angezeigt wird, daß für das Zeichnen der Titelleistenrahmen eine eigene Zeichenfunktion aufgerufen werden muß.

Siehe auch *TTinyCaption::EvNCPaint*

DoSysCommand

```
TEventStatus DoSysCommand(uint cmdType, TPoint& p);
```

Liefert *esPartial* zurück, wenn die Titelleiste nicht aktiviert ist. Wenn die Titelleiste als Symbol dargestellt ist und der Benutzer das Symbol anklickt, wird *DoSysMenu* aufgerufen, um das Menü im Normalzustand anzuzeigen und *esComplete* zurückgeliefert.

Siehe auch *TTinyCaption::EvSysCommand*

DoSysMenu

```
void DoSysMenu();
```

Holt das Systemmenü und erstellt Menüeinträge. *DoSysMenu* ist außerdem dafür verantwortlich, den Menüstatus anzuzeigen und zu verfolgen.

EnableTinyCaption

```
void EnableTinyCaption(int ch=45, bool closeBox=false);
```

Aktiviert die kleine Titelleiste. Per Voreinstellung ersetzt *EnableTinyCaption* das Systemfenster durch ein kleines Titelleistenfenster, das sich nicht schließt, wenn das Systemfenster angeklickt wird. Ist das *closeBox*-Argument **true**, bewirkt ein Klicken auf das Systemmenü das Schließen des

Fensters anstelle des Einblendends des Menüs. Wenn Sie eine kleine Titelleiste in einer abgeleiteten Klasse verwenden, können Sie das Fenster mit Hilfe von *EnableTinyCaption* verbergen. Sie können folgende Werte verwenden, um die kleine Titelleiste zu verkleinern:

```
EnableTinyCaption(30, true);
```

Folgende Werte können Sie verwenden, um die kleine Titelleiste auf maximale Größe zu bringen:

```
EnableTinyCaption(48, true);
```

EvCommand

```
LRESULT EvCommand(uint id, HWND hWndCtl, uint notifyCode);
```

EvCommand stellt zusätzliche Rechenleistung für Befehle zur Verfügung, läßt aber zunächst das zentrale Fenster und seine übergeordneten Fenster den Befehl behandeln.

Siehe auch *TTinyCaption::DoCommand*

EvLButtonUp

```
void EvLButtonUp(uint hitTest, TPoint& screenPt);
```

Reagiert durch Aufruf von *DoLButtonUp* auf eine Botschaft, die das Loslassen einer Maustaste anzeigt. Liefert *DoLButtonUp* nicht *IsComplete* zurück, ruft *EvLButtonUp TWindow::EvLButtonUp* auf.

Siehe auch *TTinyCaption::DoLButtonUp*

EvMouseMove

```
void EvMouseMove(uint hitTest, TPoint& screenPt);
```

Reagiert durch Aufruf von *DoMouseMove* auf eine Bewegung der Maus. Liefert *DoMouseMove* nicht *IsComplete* zurück, ruft *EvMouseMove TWindow::EvMouseMove* auf.

Siehe auch *TTinyCaption::DoMouseMove*

EvNCActivate

```
bool EvNCActivate(bool active);
```

Reagiert durch Aufruf von *DoNCActivate*, wenn eine Änderung einer Titelleiste oder Symbols gewünscht wird. Wenn *DoNCActivate* nicht *esComplete* zurückliefert, ruft *EvNCActivate TWindow::EvNCActivate* auf.

Siehe auch *TTinyCaption::DoNCActivate*

EvNCCalcSize

```
uint EvNCCalcSize(bool calcValidRects, NCCALCSIZE_PARAMS far& calcSize);
```

Reagiert durch Aufruf von *DoNCAvtivate*, wenn eine Änderung einer Titelleiste oder eines Symbols gewünscht wird. Liefert *DoNCActivate* nicht *esComplete* zurück, ruft *EvNCActivate TWindow::EvNCActivate* auf.

Berechnet die Größe des Befehlsfensters einschließlich der Titelleiste und des Rahmens, so daß es in das Fenster paßt.

Siehe auch *TTinyCaption::DoNCActivate*

EvNCHitTest

```
uint EvNCHitTest(TPoint& screenPt);
```

Reagiert durch Aufruf von *DoNCHitTest* auf eine Cursorbewegung oder das Drücken einer Maustaste. Wenn *DoNCHitTest* nicht *esComplete* zurückliefert, ruft *EvNCHitTest TWindow::EvNCHitTest* auf.

Siehe auch *TTinyCaption::DoNCHitTest*

EvNCLButtonDown

```
void EvNCLButtonDown(uint hitTest, TPoint& screenPt);
```

Wenn die linke Maustaste gedrückt wird und sich der Cursor zu diesem Zeitpunkt im Non-Client-Bereich der Titelleiste befindet, so reagiert *EvNCLButtonDown* darauf mit einem Aufruf von *DoNCLButtonDown*. Wenn *DoNCLButtonDown* nicht *esComplete* zurückliefert, ruft *EvNCLButtonDown TWindow::EvNCLButtonDown* auf.

Siehe auch *TTinyCaption::DoNCLButtonDown*

EvNCPaint

```
void EvNCPaint();
```

Reagiert durch Aufruf von *DoNCActivate*, wenn eine Änderung einer Titelleiste oder eines Symbols gewünscht wird. Zeichnet den angegebenen Gerätekontext oder Anzeigebildschirm und führt die besonderen Zeichenaktionen für das Zeichnen des Titels aus.

Ruft *TWindow::EvNCPaint* auf, um den angegebenen Gerätekontext oder Anzeigenbildschirm zu zeichnen.

Siehe auch *TTinyCaption::DoNCActivate*

EvSysCommand

```
void EvSysCommand(uint cmdType, TPoint& p);
```

Reagiert durch Aufruf von *DoSysCommand* auf eine WM_SYSCOMMAND-Botschaft. Wenn *DoSysCommand esPartial* zurückliefert, ruft *EvSysCommand TWindow::EvSysCommand* auf.

Siehe auch *TTinyCaption::DoSysCommand*

GetCaptionRect

```
TRect GetCaptionRect();
```

Speichert den Bereich der Titelleiste für Änderungen oder ein Neuzeichnen

Siehe auch *TTinyCap::PaintCaption*

GetMaxBoxRect

```
TRect GetMaxBoxRect();
```

Liefert die Größe des Vergrößerungssymbols zurück.

Siehe auch *TTinyCap::PaintMaxBoxRect*

GetMinBoxRect

```
TRect GetMinBoxRect();
```

Liefert die Größe des Verkleinerungssymbols zurück.

Siehe auch *TTinyCap::PaintMinBoxRect*

GetSysBoxRect

```
TRect GetSysBoxRect();
```

Liefert die Größe des Systemfensterrechtecks zurück.

Siehe auch *TTinyCap::PaintSysBoxRect*

PaintButton

```
void PaintButton(TDC& dc, TRect& boxRect, bool pressed);
```

Zeichnet eine leere Schaltfläche.

PaintCaption

```
void PaintCaption(TWindowDC &);
```

Ruft *dc.SelectObject* auf, um das gegebene Rechteck auszuwählen sowie *dc.PatBlt*, um die kleine Titelleiste unter Verwendung des für diesen Gerätekontext aktuell gewählten Pinsels zu zeichnen.

Siehe auch *TDC::SelectObject*, *TDC::PatBlt*

PaintCloseBox

```
void PaintCloseBox(TDC& dc, TRect& boxRect, bool pressed);
```

Zeichnet ein Schließsymbol auf der kleinen Titelleiste. Wenn Sie ein eigenes Schließsymbol erstellen möchten, können Sie das voreingestellte Symbol überschreiben.

Siehe auch *TTinyCap::GetSysBoxRect*

PaintMaxBox

```
void PaintMaxBox(TDC& dc, TRect& boxRect, bool pressed);
```

Zeichnet ein Vergrößerungssymbol auf der kleinen Titelleiste.

Siehe auch *TTinyCap::GetMaxBoxRect*

PaintMinBox

```
void PaintMinBox(TDC& dc, TRect& boxRect, bool pressed);
```

Zeichnet ein Verkleinerungssymbol auf der kleinen Titelleiste.

Siehe auch *TTinyCap::GetMinBoxRect*

PaintSysBox

```
void PaintSysBox(TDC& dc, TRect& boxRect, bool pressed);
```

Zeichnet die Systemfensterumrahmung.

Siehe auch *TTinyCap::GetSysBoxRect*

Antworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_WM_NCACTIVATE	EvNCActivate
EV_WM_NCCALCSIZE	EvNcCalcSize
EV_WM_NCHITTEST	EvNcHitTest
EV_WM_NCPAINT	EvNcPaint
EV_WM_NCLBUTTONDOWN	EvNclButtonDown
EV_WM_LBUTTONUP	EvLButtonUp
EV_WM_MOUSEMOVE	EvMouseMove
EV_WM_SYSCOMMAND	EvSysCommand

Klasse TToolBox toolbox.h

TToolBox, eine von *TGadgetWindow* abgeleitete Klasse, ordnet Gadgets in einer Matrix an, in der alle Spalten die gleiche Breite (die Breite des längsten Gadget) und alle Zeilen die gleiche Höhe (die Höhe des höchsten Gadget) haben.

Sie können genau angeben, wieviele Zeilen und Spalten Ihre Mauspalette enthalten soll oder *TToolBox* die Anzahl der benötigten Zeilen und Spalten berechnen lassen. Wenn Sie AS_MANY_AS_NEEDED angeben, errechnet *TToolBox* anhand der entgegengesetzten Dimension, wieviele Zeilen oder Spalten benötigt werden. Sind z. B. zwanzig Gadgets vorhanden und die von Ihnen gewünschte Spaltenzahl ist vier, so hat Ihre Matrix fünf Zeilen.

Öffentliche Konstruktoren

Konstruktor

```
TToolBox(TWindow* parent, int numColumns = 2, int numRows =
        AS_MANY_AS_NEEDED, TTileDirection direction = Horizontal,
        TModule* module = 0);
```

Erzeugt ein *TToolBox*-Objekt mit der angegebenen Anzahl von Zeilen und Spalten und der angegebenen Anordnungsrichtung. Überlappt die Begrenzungen der Mauspalette mit denen des Gadget und setzt *ShrinkWrapWidth* auf **true**.

Öffentliche Elementfunktionen

GetDesiredSize

```
void GetDesiredSize(TSize& size);
```

Überschreibt die *GetDesiredSize*-Funktion von *TGadget* und errechnet die Größe der Zelle durch Aufruf von *GetMargin*, um die Ränder zu ermitteln.

Siehe auch *TGadgetWindow::GetDesiredSize*

Insert

```
void Insert(TGadget& gadget, TPlacement = After, TGadget* sibling = 0);
```

Übergeht die *Insert*-Funktion von *TGadget* und weist den Schalter an, nicht einzurasten.

Siehe auch *TGadgetWindow::Insert*

SetDirection

```
virtual void SetDirection(TTileDirection direction);
```

Setzt eine horizontale oder vertikale Anordnungsrichtung.

Geschützte Datenelemente

NumColumns

```
int NumColumns;
```

Enthält die Anzahl der Spalten für die Toolbox.

NumRows

```
int NumRows;
```

Enthält die Anzahl der Zeilen für die Toolbox.

Geschützte Elementfunktionen

TileGadgets

```
TRect TileGadgets();
```

Ordnet die Gadgets in der angegebenen Richtung (horizontal oder vertikal) an.

Ruft *PositionGadget* auf, um abgeleiteten Klassen zu ermöglichen, die Abstände zwischen Gadgets einzustellen.

Siehe auch *TGadgetWindow::TileGadget*

Aufzählung TTransferDirection — window.h

Die *TTransferDirection*-Aufzählung beschreibt folgende Konstanten, die die Transferfunktion verwendet, um zu entscheiden, wie Daten aus dem und in den Transferpuffer übertragen werden sollen.

Tabelle 2.29: Transferfunktionskonstanten

Konstante	Bedeutung
tdGetData	Daten von der Klasse holen.
tdSetData	Daten an die Klasse senden.
tdSizeData	Größe der an die Klasse übertragenen Daten zurückliefern.

Siehe auch *TWindow::Transfer*, *TWindow::TransferData*

Klasse TUIHandle — uihandle.h

TUIHandle verwaltet und zeichnet verschiedene Arten von Oberflächen-Handles einschließlich schraffierter Rahmen-Ziehpunkte und Größenänderungsmarken (kleine Quadrate, die sich an den Kanten befinden) auf einem Rechteck. Sie können diese Klasse dazu verwenden, schraffierte Rahmen zu erzeugen, die um verschiedene Arten von Zeichenobjekten, die manipuliert werden sollen, gelegt werden können.

Mit Hilfe dieser Klasse können Sie eine Anwendung erzeugen, mit der Sie:

- die Größe des Rechtecks durch Zeigen und Greifen einer der Ziehmarken auf dem Rand verändern können
- das gesamte Rechteck durck Klicken in der Mitte des Rechtecks bewegen können

Per Voreinstellung wird ein schraffierter Rahmen mit acht Ziehpunkten erzeugt. Sie können jedoch sowohl das Vorhandensein der Ziehpunkte als auch das Muster des Rahmens verändern. Sie können auch einen gestrichelten Rahmen um ein Rechteck zeichnen oder ein Rechteck, das mit einer Schraffur gefüllt ist. *THatch8x8Brush* enthält Pinsel, mit denen Sie den schraffierten Rahmen zeichnen können.

TUIHandle verwendet den Aufzählungstyp *TWhere*, um den Bereich zurückzuliefern, in den der Anwender mit der Maus zeigt und klickt (dieser Bereich wird Aktivierungsbereich genannt).

Der folgende Programmcode aus SAMPCONT.CPP erzeugt einen schraffierten Rahmen und Oberflächen-Ziehpunkten für eine Container-Anwendung unter OLE 2.

```
// Standard-Rechteckzeichnen ausführen.
     TRect r // Fügen Sie hier Ihren Code zum Zeichnen des Rechtecks ein.
// Zeichnen von 8 Ziehpunkten mit einem Rahmen über dem Objekt.
      TUIHandle handle(r, TUIHandle::HandlesIn|TUIHandle::Grapples
      |TUIHandle::HatchBorder, 5);
     handle.Paint(dc);
// Fügen Sie hier Ihren Code ein ....
// Zeichnen eines schraffierten Rahmens.
     TUIHandle handle(r, TUIHandle::HatchBorder, 5);
     handle.Paint(dc);
```

Siehe auch *THatch8x8Brush*

Typdefinitionen

TStyle

```
enum TStyle{HandlesIn, HandlesOut, Framed, DashFramed, Grapples,
HatchBorder, HatchRect, InsideSpecial};
```

Zählt den Stil des Rahmens um das Rechteck mit Kombinationen der folgenden Stilelemente auf:

Wert	Beschreibung
Wählen Sie einen der folgenden Stile aus:	
HandlesIn	Ziehpunkte innerhalb des Rechtecks zeichnen.
HandlesOut	Ziehpunkte außerhalb des Rechtecks zeichnen.
Wählen Sie einen oder keinen der folgenden Stile aus:	
Framed	Gefüllten Rahmen um das Rechteck zeichnen.
DashFramed	Gestrichelten Rahmen um das Rechteck zeichnen.
Grapples	Acht Ziehpunkte zur Größenanpassung zeichnen.
Wählen Sie einen oder keinen der folgenden Stile aus:	
HatchBorder	Schraffierten Ziehpunkt zum Bewegen des Rahmens zeichnen.
HatchRect	Ganzes Rechteck mit Schraffur gefüllt zeichnen.
InsideSpecial	Inneren Aktivierungsbereich besonders behandeln.

Wird ein schraffierter Rahmen mit Ziehmarken innerhalb eines Rechtecks gezeichnet, sitzt dieser innerhalb der Begrenzung des äußeren Rahmen des Rechtecks. Wird ein schraffierter Rahmen mit Ziehmarken außerhalb des Rechtecks gezeichnet, wird er außerhalb der Begrenzung des Rechteckrahmens gezeichnet. Im letzten Fall gibt die Funktion *GetBoundingRect* ein größeres Rechteck zurück.

TWhere

```
enum TWhere {TopLeft, TopCenter, TopRight, MidLeft, MidCenter, MidRight,
BottomLeft, BottomCenter, BottomRight, Outside, Inside};
```

TWhere gibt an, welcher der Ziehpunkte oder Handles aktiviert wurde. *TWhere* kann einen der folgenden Werte enthalten:

Wert	Beschreibung
TopLeft	Der Ziehpunkt in der oberen linken Ecke des Rahmens wurde aktiviert.
TopCenter	Der Ziehpunkt in der oberen Mitte des Rahmens wurde aktiviert.
TopRight	Der Ziehpunkt in der oberen rechten Ecke des Rahmens wurde aktiviert.
MidLeft	Der Ziehpunkt in der Mitte des linken Rahmens wurde aktiviert.
MidCenter	Der Bereich in der Mitte des Rechtecks wurde aktiviert. Dies bedeutet, daß entweder der schraffierte Rahmen oder der innere Bereich aktiviert wurden.
MidRight	Der Ziehpunkt in der Mitte des rechten Rahmens wurde aktiviert.
BottomLeft	Der Ziehpunkt in der linken unteren Ecke des Rahmens wurde aktiviert.
BottomCenter	Der Ziehpunkt in der unteren Mitte des Rahmens wurde aktiviert.
BottomRight	Der Ziehpunkt in der rechten unteren Ecke des Rahmens wurde aktiviert.
Outside	Der Aktivierungsbereich liegt vollständig außerhalb des Objekts.
Inside	Der Aktivierungsbereich für den Ziehpunkt liegt innerhalb des Objekts und *InsideSpecial* ist gesetzt.

Das Setzen von *InsideSpecial* bezieht sich auf den Bereich innerhalb des Rechtecks, wenn der Aktivierungsbereich besonders behandelt werden muß, da er beispielsweise Text oder Grafik enthält. Normalerweise bedeutet eine Aktivierung des Bereichs innerhalb des Rechtecks, daß der Anwender das Rechteck bewegen möchte. Enthält das Rechteck jedoch Text, klickt der Anwender vielleicht auch in diesen Bereich, um Text einzugeben. Dieser Sachverhalt wird als ein Fall von *inside special* bezeichnet.

Der Aktivierungsbereich (Where) kann unter Verwendung der folgenden Gleichung in Zeilen und Spalten umgewandelt werden:

Zeile = Where / 3

Spalte = Where mod 3

Anschließend können Sie diese Werte verwenden, um die Bewegung des Objekts zu berechnen und um das Objekt in der Größe zu verändern.

Öffentlicher Konstruktor

Konstruktor

```
TUIHandle(const TRect& frame, uint style =
HandlesIn|Grapples|HatchBorder, int thickness = 5);
```

Erzeugt ein *TUIHandle*-Objekt mit acht Ziehmarken in einem schraffierten Rahmen sowie einer voreingestellten Dicke von fünf Pixeln nach innen für den angegebenen Rahmen.

Öffentliche Elementfunktionen

GetBoundingRect

```
GetBoundingRect() const;
```

GetBoundingRect gibt ein Rechteck mit gemäß der Dicke angepaßter Größe zurück. Wenn beispielsweise die Ziehpunkte außerhalb des Rechtecks liegen, liefert *GetBoundingRect* ein größeres Rechteck zurück. Der Aufzählungstyp *TStyle* definiert die Position der Ziehmarken als *HandlesIn* oder *HandlesOut*.

GetCursorId

```
static uint16 GetCursorId(TWhere where);
```

Liefert die ID eines Standard-Cursors zurück, der für den Einsatz in dem durch den Parameter *where* angegebene Bereich geeignet ist.

HitTest

```
TWhere HitTest(const TPoint& point) const;
```

Vergleicht einen gegebenen Punkt (*point*) mit verschiedenen anderen Punkten des Rechtecks. Lag die Aktivierung außerhalb des Rechtecks, liefert *HitTest Outside* zurück. Wurde der schraffierte Rahmen-Ziehpunkt des Rechtecks aktiviert, wird *MidCenter* (innen) zurückgeliefert. Für alle anderen Aktivierungen liefert *HitTest* die Ziehpunktposition, die aktiviert wurde, zurück. Der Aufzählungstyp *TWhere* definiert die möglichen Aktivierungsbereiche.

Move

```
void Move(int dx, int dy);
```

Bewegt das Rechteck relativ zu den in *dx* und *dy* angegebenen Werten.

Siehe auch *TUIHandle::MoveTo*

MoveTo

```
void MoveTo(int x, int y);
```

Bewegt das Rechteck zu den angegebenen Koordinaten *x* und *y*.

Siehe auch *TUIHandle::Move*

Paint

```
void Paint(TDC& dc) const;
```

Zeichnet das *TUIHandle*-Objekt in den angegebenen Gerätekontext *dc*.

Size

```
void Size(int w, int h);
```

Setzt die Größe des Rechtecks gemäß den Werten in *w* (Breite) und *h* (Höhe).

Klasse TValidator validate.h

TValidator, eine streamfähige Klasse, definiert ein abstraktes Objekt zur Datenprüfung. Obgleich Sie niemals wirklich eine Instanz von *TValidator* erstellen werden, stellt es die abstrakten Funktionen für die anderen Datenprüfungsobjekte zur Verfügung.

Das Beispielprogramm VALIDATE.CPP auf Ihrer Diskette leitet auf folgende Weise *TVAlidateApp* von *TApplication* ab:

```
class TValidateApp : public TApplication {
 public:
   TValidateApp() : TApplication("ValidateApp") {}
   void InitMainWindow() {
     MainWindow = new TTestWindow(0, "Validate Dialog Input");
 }
```

und zeigt ein Meldungsfenster an, wenn der Benutzer eine ungültige Angestellten-ID eingegeben hat.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TValidator();
```

Erstellt ein abstraktes Prüfungsobjekt und setzt *Options*-Felder auf 0.

Destructor

```
virtual ~TValidator() {}
```

Entfernt ein abstraktes Prüfungsobjekt.

Öffentliche Elementfunktionen

Error

```
virtual void Error();
```

Error ist eine abstrakte Funktion, die von der Funktion *Valid* aufgerufen wird, wenn sie feststellt, daß der Anwender ungültige Informationen eingegeben hat. Per Voreinstellung führt *TValidator::Error* nichts aus, abgeleitete Klassen können *Error* jedoch übergehen, um dem Benutzer Rückmeldungen zu liefern.

HasOption

```
inline bool HasOption(int option);
```

Speichert *Options*-Bits. Liefert **true** zurück, wenn eine festgelegte Option gesetzt wird.

Siehe auch *TValidator::Options*, *Voxxxx*-Konstanten

IsValid

```
virtual bool IsValid(const char far* str);
```

Per Voreinstellung liefert *TValidator:IsValid* **true** zurück. Abgeleitete Prüfungsklassen können *IsValid* übergehen, um Daten für ein fertiggestelltes Editierfeld zu prüfen. Besitzt ein Editierfeld ein zugehöriges Prüfungsob-

jekt, ruft die *Valid*-Methode des ersteren die *Valid*-Methode des Prüfungsobjekts auf, welche ihrerseits *IsValid* aufruft, um zu entscheiden, ob die Inhalte des Editierfelds gültig sind.

Siehe auch *TValidator:Valid*

IsValidInput

```
virtual bool IsValidInput(char far* str, bool suppressFill);
```

Ist ein Editierfeld mit einem Prüfungsobjekt verknüpft, so ruft es nach jedem Tastaturereignis *IsValidInput* auf. Dies ermöglicht Prüfungsobjekten, wie z.B. Filter-Validatoren, Fehler zu erkennen, noch ehe der Benutzer seine vollständige Eingabe gemacht hat.

Per Voreinstellung liefert *IsValidInput* **true** zurück. Abgeleitete Datenprüfer können *IsValidInput* umgehen, um die Gültigkeit von Daten noch während der Eingabe durch den Benutzer zu überprüfen. Solche Validatoren liefern TRUE zurück, wenn *str* gültige Daten enthält; ansonsten **false**.

str ist der aktuelle Eingabestring. *suppressFill* entscheidet, ob das Prüfungsobjekt den String vor der Überprüfung automatisch formatieren soll. Ist *suppressFill* **true**, findet die Überprüfung am unveränderten String *str* statt. Wenn *suppressFill* **false** ist, soll der Validator vor der Datenüberprüfung jede mögliche Füllform anwenden. Von den Standard-Prüfungsobjekten prüft nur *TPXPictureValidator* den Wert von *suppressFill*.

IsValidInput kann den Inhalt des Eingabestrings verändern; es kann z.B. alle Zeichen in Versalien setzen oder korrekte Zeichen aus einer Formatmaske einfügen, jedoch sollte *IsValidInput* keine ungültigen Zeichen aus dem String löschen. Durch Rückgabe von **false** zeigt *IsValidInput* an, daß das Editierfeld die falschen Zeichen entfernen soll.

SetOption

```
inline void SetOption(int option);
```

Setzt die Bits für das *Options*-Datenelement.

Siehe auch *TValidator::Options*, *Voxxxx*-Konstanten

Transfer

```
virtual uint Transfer(char far* str, void* buffer, TTransferDirection
                      direction);
```

Ermöglicht einem Prüfungsobjekt das Setzen und Lesen von Daten im zugehörigen Editierfeld. Dies ist vornehmlich für Prüfungsobjekte nützlich, die Non-String-Daten wie etwa numerische Werte überprüfen. *TRangeValidator* verwendet z. B. *Transfer*, um Werte zu lesen und zu schreiben, anstatt einen vollständigen String zu übertragen.

Per Voreinstellung geben Editierfelder mit Prüfungsobjekten dem Validator durch Aufruf seiner *Transfer*-Methode eine erste Möglichkeit, auf *DataSize*, *GetData* und *SetData* zu reagieren. Liefert *Transfer* irgendetwas anderes als 0 zurück, zeigt es dem Editierfenster an, daß es den zugehörigen Transfer behandelt hat. Die Vorgabehandlung für *TValidator:Transfer*

ist, stets 0 zurückzuliefern. Sie müssen, wenn Sie das Prüfungsobjekt Daten übertragen lassen möchten, seine *Transfer*-Methode aufheben.

Die ersten beiden Parameter von *Transfer* sind der Textstring im Editierfeld und der *tdGetData*- oder *tdSetData*-Datensatz. Abhängig vom Wert von *direction* kann *Transfer* entweder *str* aus *buffer* setzen oder die Daten aus *str* in *buffer* einlesen. Rückgabewert ist stets die Anzahl der übertragenen Bytes.

Ist *direction tdSizeData*, ändert *Transfer* weder *str* noch *buffer*, sondern gibt einfach die Größe der Daten zurück. Wenn *direction tdSetData* ist, wandelt *Transfer str* in den entsprechenden Datentyp um und schreibt, unter Rückgabe der Anzahl der geschriebenen Bytes, den Wert in *buffer*.

Siehe auch Aufzählung *TTransferDirection*

UnsetOption

```
inline void UnsetOption(int option);
```

Löscht die Bits, die denen im Element *options* spezifizierten entsprechen.

Siehe auch *TValidator::Options*, *Voxxxx*-Konstanten

Valid

```
bool Valid(const char far* str);
```

Liefert **true** zurück, wenn *IsValid* TRUE zurückliefert. Andernfalls wird *Error* aufgerufen und **false** zurückgeliefert. Die *Valid*-Methode eines Prüfungsobjekts wird von der *Valid*-Methode seines zugehörigen Editierfelds aufgerufen.

Editierfelder mit verknüpften Prüfungsobjekten rufen die *Valid*-Methode des Prüfungsobjekts unter zwei Bedingungen auf. Die erste ist, daß die *ofValidate*-Option des Editierfelds gesetzt ist und das Editierfeld *Valid* aufruft, wenn es den Fokus verliert. Die zweite Bedingung ist, daß das Dialogfenster, welches das Editierfeld enthält, *Valid* für all seine Dialogelemente aufruft, normalerweise weil der Benutzer das Dialogfenster schließen möchte oder einen Eingabebildschirm akzeptieren will.

Geschützte Datenelemente

Options

```
uint16 Options;
```

Options ist ein Bitmap-Element, das zur Steuerung von Optionen verschiedener Nachkommen von *TValidator* verwendet wird. Per Voreinstellung löscht der *TValidator*-Konstruktor alle in *Options* enthaltenen Bits.

Siehe auch: *voxxxx*-Konstanten, *TValidator::SetOption*, *TValidator::UnsetOption*

Klasse TValidator::TXValidator validate.h

TXValidator, eine geschachtelte Klasse, beschreibt einen aus eine ungültigen Prüfungsobjekt resultierenden Exception.

Öffentliche Konstruktoren

Konstruktor

```
TXValidator(uint resId = IDS_VALIDATORSYNTAX );
```

Erstellt ein *TXValidator*-Objekt, wobei die Ressourcen-ID auf IDS_VALIDATORSYNTAX gesetzt wird.

Klasse TVbxControl vbxctl.h

TVbxControl, eine von *TControl* abgeleitete Klasse, liefert die Schnittstelle für VisualBasic(VBX)-Elemente. Sie können diese Klasse dazu verwenden, um Eigenschaften von VBX-Dialogelementen zu erfragen oder festzulegen. Unter bestimmten Bedingungen können Sie auch zusätzliche Methoden zur Verarbeitung von Dialogelementen verwenden.

Sie können die Eigenschaften des Dialogelements unter Verwendung eines Indexwertes oder eines Namens manipulieren. Es stehen verschiedene überladene *GetProp*-Funktionen zur Verfügung, so daß Sie Zugriff auf verschiedene Typen von Eigenschaften haben. Auf ähnliche Weise können Sie die Eigenschaften von Dialogelementen über verschiedene überladene *SetProp*-Funktionen bestimmen, indem Sie entweder den Namen des VBX-Elements oder den Indexwert verwenden. Den Namen der Eigenschaft, die Sie manipulieren wollen, finden Sie in der Dokumentation zu den VBX-Elementen.

Das Beispielprogramm VBXCTLX.CPP befindet sich auf Ihrer Beispieldiskette.

TVbxControl stellt Funktionen zur Ereignisbehandlung zur Verfügung, die Anfragen zum Vergleichen, Löschen, Zeichnen oder Messen der VBX-Elemente verarbeiten. Wenn das TVbx-Dialogelement VBX-Ereignisse verarbeiten soll, müssen Sie eine Klasse von *TVbxControl* ableiten und ihr eine Antworttabelle mit Einträgen für die Ereignisse, die bearbeitet werden sollen, beifügen. Informationen über die Erstellung von Antworttabellen und die Handhabung von VBX-Botschaften finden Sie unter *TVbxEventHandler*.

Öffentliche Konstruktoren und Destruktor

```
TVbxControl(TWindow* parent, int id, const char far* vbxName, const char
far* vbxClass, const char far* title, int x, int y, int w, int h, long
initLen = 0, void far* initData = 0, TModule* module = 0);
```

Erzeugt ein VBX-Element, wobei *parent* auf das übergeordnete Fenster zeigt, *id* die ID des Dialogelements repräsentiert, *vbxName* den Namen der das VBX-Element enthaltenden Datei, *vbxClass* den Namen der VBX-Klasse, *title* die Titelleiste des Elements. *x* und *y* sind die Koordinaten für den Punkt im übergeordneten Fenster, an dem das Dialogelement plaziert werden soll, *w* und *h* die Breite und Höhe des Dialogelements und *module* ist die Bibliotheksressourcen-ID für das Dialogelement.

Konstruktor

```
TVbxControl(TWindow* parent, int resourceId, TModule* module = 0);
```

Ist ein VBX-Dialogelement Teil einer Dialogressource, kann seine ID dazu verwendet werden, ein entsprechendes (oder Alias-) ObjectWindows-Objekt zu erstellen. Sie können diesen Konstruktor verwenden, wenn ein VBX-Element bereits in der Ressourcendatei der Anwendung definiert wurde. *resourceId* ist die Ressource-ID des VBX-Elements in der Ressourcendatei.

Destruktor

```
~TVbxControl();
```

Löscht das *TVbxControl*-Objekt.

Öffentliche Elementfunktionen

AddItem

```
bool AddItem(int index, const char far* item);
```

Fügt der Liste von VBX-Dialogelementen ein Element (*item*) unter dem spezifizierten Index (*index*) hinzu. Bei Erfolg wird ein Wert ungleich Null zurückgeliefert.

Drag

```
bool Drag(int action);
```

Steuert den Ziehen- und Ablegen-Status des VBX-Elements gemäß dem Wert von *action*, der 0 (Abbrechen eines Ziehen-Vorgangs), 1 (mit dem Ziehen eines Dialogelements beginnen) oder 2 (Ziehen beenden) sein kann.

GetEventIndex

```
int GetEventIndex(const char far* name);
```

Liefert den Index des Ereignisses zurück, das mit dem in *name* übergebenen Ereignisses verknüpft ist. Tritt ein Fehler auf, wird -1 zurückgeliefert.

GetEventName

```
const char far* GetEventName(int eventindex);
```

Liefert einen String zurück, der den Namen eines Ereignisses enthält, das mit der Indexzahl (*eventindex*) des Integer-Ereignisses verknüpft ist. Tritt ein Fehler auf, wird 0 zurückgeliefert.

GetHCTL

```
HCTL GetHCTL()
```

Liefert ein Handle zu einem mit dem vorliegenden *TVbxControl*-Objekt verknüpften VBX-Element zurück.

GetNumEvents

```
int GetNumEvents();
```

Liefert die Gesamtzahl der mit dem VBX-Element verknüpften Ereignisse zurück.

GetNumProps

```
int GetNumProps();
```

Liefert die Gesamtzahl der mit dem VBX-Element verknüpften Eigenschaften zurück.

GetProp

```
bool GetProp(int propIndex, int& value, int arrayIndex = -1);
```

Ermittelt den Integerwert einer Eigenschaft. *GetProp*, eine überladene Funktion, ermittelt eine VBX-Elementeigenschaft unter Verwendung eines Indexwertes. *propIndex* ist der Indexwert der Eigenschaft, deren Wert Sie ermitteln wollen. *value* ist eine Referenz zu der Variablen, die die Eigenschaftswerte empfängt. *arrayIndex*, ein optionales Argument, bestimmt die Position des Werts in einer Array-Eigenschaft, wenn es sich bei der Eigenschaft um einen Arraytyp handelt. Ist die Eigenschaft kein Arraytyp, nimmt *arrayIndex* gemäß Voreinstellung den Wert -1 an. Im Referenzhandbuch zu den VBX-Elementen finden Sie eine Beschreibung, wie der Datenyp einer Eigenschaft ermittelt wird. Bei Erfolg liefert *GetProp* Nichtnull zurück. Verwenden Sie, um die Eigenschaft über die Spezifizierung des Eigenschaftsindex zu ermitteln, eine der folgenden sechs *GetProp*-Funktionen.

GetProp

```
bool GetProp(int propIndex long& value, int arrayIndex = -1);
```

Speichert den **long**-Wert einer Eigenschaft.

GetProp

```
bool GetProp(int propIndex, ENUM& value, int arrayIndex= -1);
```

Speichert einen aufgelisteten Eigenschaftswert. Als Aufzählungstyp könnte beispielsweise eine mit einer bestimmten Schriftart verknüpfte Liste von Optionen definiert sein.

GetProp

```
bool GetProp(int propIndex, HPIC& value, int arrayIndex=-1);
```

Speichert eine Maske (*value*). *HPIC* ist das Handle zu dieser Maske.

GetProp

```
bool GetProp(int propIndex float& value, int arrayIndex = -1);
```

Speichert den Fließkommawert einer Eigenschaft.

GetProp

```
bool GetProp(int propIndex string& value, int arrayIndex = -1);
```

Speichert den Stringwert einer Eigenschaft.

GetProp

```
bool GetProp(int propIndex COLORREF value, int arrayIndex = -1);
```

Speichert den Farbwert einer Eigenschaft.

GetProp

```
bool GetProp(const char far* name, int& value, int arrayIndex = -1);
```

Liefert einen Integer-Datenwert zurück. *GetProp*, eine überladene Funktion, speichert die Eigenschaft eines VBX-Dialogelements. *propIndex* ist der Indexwert der Eigenschaft, deren Wert Sie ermitteln wollen. *value* ist eine Referenz zu der Variable, die den Eigenschaftswert empfängt. *arrayIndex*, ein optionales Argument, bestimmt die Position des Werts in einer Array-Eigenschaft, wenn es sich bei der Eigenschaft um einen Arraytyp handelt. Ist die Eigenschaft kein Arraytyp, nimmt *arrayIndex* gemäß Voreinstellung den Wert -1 an. Bei Erfolg liefert *GetProp* einen Wert ungleich Null zurück. Verwenden Sie, um die Eigenschaft über die Spezifizierung des Eigenschaftnamens zu ermitteln, eine der folgenden sechs *GetProp*-Funktionen.

GetProp

```
bool GetProp(const char far* name, long& value, int arrayIndex = -1);
```

Speichert den **long**-Wert einer Eigenschaft.

GetProp

```
bool GetProp(const char far* name, float& value, int arrayIndex = -1);
```

Speichert den Fließkommawert einer Eigenschaft.

Getprop

```
bool Getprop(const char far* name, ENUM& value, int arrayIndex=-1);
```

Speichert den Aufzählungswert einer Eigenschaft.

Getprop

```
bool Getprop(const char far* name, HPIC& value, int arrayIndex=-1);
```

Speichert den Eigenschaftswert einer Maske (*value*). *HPIC* ist ein Handle zu dieser Maske.

GetProp

```
bool GetProp(const char far* name, string& value, int arrayIndex = -1);
```

Speichert den Stringwert einer Eigenschaft.

GetProp

```
bool GetProp(const char far* name, COLORREF value, int arrayIndex = -1);
```

Speichert den Stringfarbwert einer Eigenschaft.

Siehe auch *TVbxControl::SetProp*

GetPropIndex

```
int GetPropIndex(const char far* name);
```

Ermittelt den Integer-Indexwert für den in *name* übergebenen Eigenschaftsnamen. Tritt ein Fehler auf, wird -1 zurückgeliefert. Dies zeigt normalerweise an, daß der in *name* übergebene Eigenschaftsname nicht gefunden werden konnte.

GetPropName

```
const char far* GetPropName(int propIndex);
```

Ermittelt den Namen des in *index* übergebenen Eigenschaftsindex'. Bei Auftreten eines Fehlers wird 0 zurückgeliefert.

GetPropType

```
int GetPropType(int propIndex);
int GetPropType(char far* name);
```

Die erste Form ermittelt den Typ der durch *index* spezifizierten Eigenschaft. Die zweite Form ermittelt den Stringtyp der durch *name* spezifizierten Eigenschaft. Tritt ein Fehler auf, so wird 0 zurückgeliefert. Die

folgende Tabelle zeigt eine Liste der Eigenschaftstypen und ihrer entsprechenden C++ Datentypen.

Tabelle 2.30: Eigenschafts- und C++Typen

Eigenschaftstyp	C++ Typ
PTYPE_CSTRING	HSZ
PTYPE_SHORT	short
PTYPE_LONG	int32
PTYPE_BOOL	bool
PTYPE_COLOR	uint32 oder COLORREF
PTYPE_ENUM	uint8 oder ENUM
PTYPE_REAL	float
PTYPE_XPOS	int32 (Twips)
PTYPE_XSIZE	int32 (Twips)
PTYPE_YPOS	int32 (Twips)
PTYPE_YSIZE	int32 (Twips)
PTYPE_PICTURE	HPIC
PTYPE_BSTRING	HLSTR

IsArrayProp

```
bool IsArrayProp(int propIndex);
bool IsArrayProp(char far* name);
```

Liefert **true** zurück, wenn die durch *index* (erste Form) oder *name* (zweite Form) spezifizierte Eigenschaft eine Array-Eigenschaft ist.

Method

```
bool Method(int method, long far* args);
```

Method, das verwendet wird, um angepaßte Methoden zu invozieren, liefert **true** zurück, wenn ein VBX-Element auf die spezifizierte Methode (*method*) antworten kann.

Move

```
bool Move(int x, int y, int w, int h);
```

Bewegt ein VBX-Element zu dem durch die Bildschirmkoordinaten *x* und *y* bestimmten Punkt, der die obere linke Ecke des Bildschirms markiert. Legt die Größe des VBX-Elements mit *w* Pixeln in der Breite und *h* Pixeln in der Höhe neu fest. Bei Erfolg wird ein Wert ungleich Null zurückgeliefert.

Refresh

```
bool Refresh();
```

Zeichnet den Anzeigebereich des Dialogelements neu.

RemoveItem

```
bool RemoveItem(int index);
```

Entfernt ein Element (spezifiziert durch *index*). Das Element könnte beispielsweise aus einem Listenfeld, einem Kombinationsfeld oder einer Datenbank entfernt werden.

SetProp

```
bool SetProp(int propIndex, int value, int arrayIndex = -1);
```

Setzt die Eigenschaft auf einen Integerwert. *SetProp*, eine überladene Funktion, setzt eine VBX-Elementeigenschaft. *propIndex* ist die Indexzahl der Eigenschaft, deren Wert Sie setzen wollen. *value* spezifiziert den neuen Wert für

die Eigenschaft, *arrayIndex* die Position des Wertes in einer Array-Eigenschaft, wenn es sich bei der Eigenschaft um einen Arraytyp handelt. Ist die Eigenschaft kein Arraytyp, nimmt *arrayIndex* gemäß Voreinstellung den Wert -1 an. Verwenden Sie, um die Eigenschaft durch Übergeben ihres Indexwertes zu setzen, eine der folgenden sieben *SetProp*-Funktionen.

SetProp

```
bool SetProp(int propIndex, long value, int arrayIndex = -1);
```

Setzt die Eigenschaft auf einen **long**-Wert.

SetProp

```
bool SetProp(int propIndex, ENUM value, int arrayIndex= 1);
```

Setzt die Eigenschaft auf einen Aufzählungswert.

SetProp

```
bool SetProp(int propIndex, HPIC value, int arrayIndex= 1);
```

Setzt eine Maske auf einen HPIC- oder Maskenwert.

SetProp

```
bool SetProp(int propIndex, float value, int arrayIndex = -1);
```

Setzt die Eigenschaft auf einen Fließkommawert.

SetProp

```
bool SetProp(int propIndex, const string& value, int arrayIndex = -1);
```

Setzt die Eigenschaft auf einen Stringwert.

SetProp

```
bool SetProp(int propIndex, const char far* value,
                    int arrayIndex = -1);
```

Setzt die Eigenschaft auf einen Zeichenstringwert.

SetProp

```
bool SetProp(int propIndex, COLORREF value, int arrayIndex = -1);
```

Setzt die Eigenschaft auf einen Farbwert.

SetProp

```
bool SetProp(const char far* name, int value, int arrayIndex = -1);
```

Setzt die Eigenschaft auf einen Integerwert. *SetProp*, eine überladene Funktion, setzt eine VBX-Elementeigenschaft. *arrayIndex* spezifiziert die Position des Wertes in einer Array-Eigenschaft, wenn es sich bei der Eigenschaft um einen Arraytyp handelt. Ist die Eigenschaft kein Arraytyp, nimmt *arrayIndex* gemäß Voreinstellung den Wert -1 an. Verwenden Sie, um die Eigenschaft unter Verwendung ihres Namens zu setzen, eine der folgenden sieben *SetProp*-Funktionen.

SetProp

```
bool SetProp(const char far* name, long value, int arrayIndex = -1);
```

Setzt die Eigenschaft auf einen **long**-Wert.

SetProp

```
bool SetProp(int propIndex, ENUM value, int arrayIndex=-1);
```

Setzt die Eigenschaft auf einen Aufzählungswert.

SetProp

```
bool SetProp(int propIndex, HPIC value, int arrayIndex = -1);
```

Setzt die Maskeneigenschaft auf einen HPIC- oder Maskenwert.

SetProp

```
bool SetProp(const char far* name, float value, int arrayIndex = -1);
```

Setzt die Eigenschaft auf einen Fließkommawert.

SetProp

```
bool SetProp(const char far* name, const string& value,
                    int arrayIndex = -1);
```

Setzt die Eigenschaft auf einen Stringwert.

SetProp

```
bool SetProp(const char far* name, const char far* value,
                    int arrayIndex = -1);
```

Setzt die Eigenschaft auf einen Zeichenstring-Wert.

SetProp

```
bool SetProp(const char far* name, COLORREF value, int arrayIndex = -1);
```

Setzt die Eigenschaft auf einen Farb-Stringwert.

Siehe auch *TVbxControl::GetProp*

Geschützte Elementfunktionen

GetClassName

```
char far* GetClassName();
```

Ermittelt den Namen der VBX-Fensterklasse.

GetVBXProperty

```
bool GetVBXProperty(int propIndex, void far* value, int arrayIndex = -1);
```

Liefert einen Wert ungleich Null zurück, wenn die spezifizierte Eigenschaft exisitiert. *propIndex* spezifiziert den Indexwert der Integereigenschaft, deren Wert Sie ermitteln möchten. *value* zeigt auf die Variable, in der der Wert gespeichert wird.

PerformCreate

```
void PerformCreate(int menuOrId);
```

Erzeugt ein neues Dialogelement-Fenster und verknüpft das VBX-Element mit diesem Fenster. Erstellt die Kontroll-ID, Namen und Klasse des VBX-Elements sowie die Titelleiste. Setzt *Attr.style* auf den Fensterstil des Dialogelements, *Attr.X* und *Attr.Y* auf die Bildschirmkoordinaten der oberen linken Ecke des Dialogelements und *Attr.W* und *Attr.H* auf Breite und Höhe des Dialogelements.

SetVBXProperty

```
bool SetVBXProperty(int propIndex, LONG value, int arrayIndex=-1);
```

Liefert Nichtnull zurück, wenn der spezifizierte Eigenschaftswert gesetzt wurde, oder 0 bei Mißerfolg. *propindex* ist die Indexzahl der Eigenschaft, deren Wert Sie setzen wollen, *value* der zu speichernde Wert. *arrayIndex* ist ein optionales Argument, das auf -1 voreingestellt ist und den Indexwert in einem Array oder Werte der zu setzenden Eigenschaft bestimmt.

Antworttabelleneinträge

Die Klasse *TVbxControl* hat keine Antworttabelleneinträge.

Klasse TVbxEventHandler vbxctl.h

TVbxEventHandler, eine aus *TEventHandler* abgeleitete Klasse, behandelt von VBX-Elementen ausgelöste Ereignisse. Obwohl Sie diese Klasse nie werden modifizieren müssen, muß *TVbxEventHandler* mit in Ihre Fensterklasse aufgenommen werden, so daß es Ereignisse von VBX-Elementen empfangen kann. Ein Beispiel:

```
class TMyWindow:public TWindow, public TVbxEventHandler
{
// Hier Klassendefinition einfügen.
}
```

Wenn ein VBX-Element eine Ereignisbotschaft absendet, vollzieht sich folgende Ereigniskette:

1. Das VBX-Element sendet eine WM_VBXFIREEVENT-Botschaft an *TMyWindow*.
2. Der *TVbxEventHandler* von *TMyWindow* findet eine WM_VBXFIREEVENT-Botschaft in seiner Antworttabelle und ruft *EvVbxDispatch* auf.
3. Ist ein untergeornetes Fenster vorhanden, sendet *EvVbxDispatch* das Ereignis dorthin.
4. Befindet sich in der Antworttabelle des untergeordneten Fensters eine Funktion für die Ereignisbehandlung, bearbeitet das untergeordnete Fenster das Ereignis.
5. Gibt es kein untergeordnetes Fenster oder bearbeitet das untergeordnete Fenster das Ereignis nicht, sendet *EvVbxDispatch* das Ereignis an die Antworttabelle von *TMyWindow*.

Mit anderen Worten, wenn ein VBX-Element eine WM_VBXFIREEVENT-Botschaft abschickt, empfängt sie zunächst der *TVbxEventHandler* des übergeordneten Fensters, wandelt sie in eine für eine Fenster-Antworttabelle verständliche Form um und versucht die umgewandelte Botschaft an das untergeordnete Fenster zu senden. Gibt es kein solches untergeordnetes Fenster oder bearbeitet dieses die Botschaft nicht, so sendet *TVbx EventHandler* die umgewandelte Botschaft an das übergeordnete Fenster. Empfängt das übergeordnete Fenster die Botschaft, ruft es die zur Botschaft passende Handler-Funktion auf.

Zwei Antworttabellenmakros, EV_VBXEVENTNAME und EV_VBXEVENTINDEX, verweisen VBX-Ereignisse an Handler-Funktionen. Von diesen beiden Makros ist EV_VBXEVENTNAME das gebräuchlichere. EV_VBXEVENTINDEX ist zur Verwendung mit Erzeugern von Programmcodes gedacht, die die Ereignisindexwerte für ein VBX-Element

bestimmen können. Beide Makros rufen eine Ereignishandler-Funktion auf und zeigen auf die VBXEVENT-Struktur. Ein typischer Eintrag in eine EV_VBXEVENTNAME-Antworttabelle könnte etwa lauten:

```
EV_VBXEVENTNAME(IDC_BUTTON1, "MouseMove", EvMouseMove);
```

wobei IDC_BUTTON1 die Ereignis-ID ist, »EvMouseMove« der Ereignisname und *EvMouseMove* die Handler-Funktion.

Der L-Parameter einer WM_VBX FIREEVENT-Botschaft zeigt auf eine VBXEVENT-Struktur, in der Informationen über das Ereignis und das Dialogelement, welches das Ereignis erzeugt hat, gespeichert sind. Die VBXEVENT-Struktur enthält folgende Elemente:

```
typedef struct VBXEVENT {
          HCTL    Control;
          HWND    Window;
          int     ID;
          int     EventIndex;
          LPCSTR  EventName;
          int     NumParams;
          LPVOID  ParamList;
} VBXEVENT;
```

wobei

- *Control* ein Handle für das VBX-Element ist, das die Botschaft sendet.
- *Window* das Handle des Fensters des VBX-Elements ist.
- *ID* die ID des VBX-Elements ist.
- *EventIndex* der Ereignisindex ist.
- *EventName* der Name des Ereignisses ist.
- *NumParams* die Anzahl der Ereignisargumente speichert.
- *ParamList* ein Zeiger auf eine Liste von Zeigern auf die einzelnen Argumente des Ereignisses ist. Das Datenelement *ParamList* ermöglicht Zugriff auf die aktuellen Argumente des Ereignisses.

Um VBX-Ereignisse zu bearbeiten, verwendet Ihr Programm eine Funktion für die Ereignisbehandlung. Im folgenden Beispiel ist *EvMouseMove* der Name der Handler-Funktion, die einen Zeiger an die VBXEVENT-Ereignisstruktur übergibt.

VBX_EVENTARGNUM ist das Makro, welches als Parameter *event*, einen Typ und eine Indexzahl besitzt. *event* verweist auf die VBXEVENT-Struktur, **short** ist der Argumenttyp des Ereignisses und 0 und 1 sind die Indexzahlen des Arguments. Im Handbuch zum VBX-Element finden Sie Argumenttypen und Indizes.

```
void EvMouseMove(VBXEVENT FAR* event)
{
   short X = VBX_EVENTARGNUM(event, short, 0);
```

```
    short Y = VBX_EVENTARGNUM(event, short, 1);
}
```

Da VBX-Elemente ursprünglich zur Verwendung mit Visual Basic entworfen wurden, sind ihre Ereignisargumente mit Basic-Datentypen belegt. Die folgende Tabelle listet die Basic-Typen, ihre Entsprechungen in C++ sowie die Makros auf.

Tabelle 2.31: Basic und C++ VBX-Datentypen

Basic	C++	Makro
Boolean	short	VBX_EVENTARGNUM(*event*, *short*, *index*)
Control	HCTL	VBX_EVENTARGNUM(*event*, *HCTL*, *index*)
Double	double	VBX_EVENTARGNUM(*event*, *double*, *index*)
Enum	short	VBX_EVENTARGNUM(*event*, *short*, *index*)
Integer	short	VBX_EVENTARGNUM(*event*, *short*, *index*)
Long	long	VBX_EVENTARGNUM(*event*, *long*, *index*)
Single	float	VBX_EVENTARGNUM(*event*, *float*, *index*)
String	HLSTR	VBX_EVENTARGSTR(*event*, *index*)

Die folgende Tabelle listet die VBX-Standardereignisse und die entsprechenden von der Borland C++ VBX-Emulationsbibliothek unterstützten Argumente auf.

Tabelle 2.32: VBX-Ereignisargumente

Ereignis	Argumente
Click	Keine
DblClick	Keine
DragDrop	Source als Control, X als Integer, Y als Integer
DragOver	Source als Control, X als Integer, Y als Integer, State als Integer
GotFocus	Keine
KeyDown	Key als Integer, Shift als Integer
KeyPress	Key als Integer, Shift als Integer
KeyUp	Key als Integer, Shift als Integer
LostFocus	Keine
MouseDown	X als Integer, Y als Integer
MouseMove	X als Integer, Y als Integer, Shift als Integer, Button als Integer
MouseUp	X als Integer, Y als Integer, Shift als Integer, Button als Integer

Das Statusargument für das DragOver-Ereignis kann einen der folgenden Werte haben:

- 0, wobei das Quellelement innerhalb eines Zielbereichs gezogen wird.
- 1, wobei das Quellelement außerhalb eines Zielbereichs gezogen wird.
- 2, wobei das Quellelement von einer Position des Zielbereichs auf eine andere bewegt wird.

Sowohl für das DragOver- als auch für das DragDrop-Ereignis sollte der Control-Argumenttyp für C++ nach HCTL (einem Handle für das VBX-Dialogelement) übertragen werden. Die X- und Y-Werte sind in Pixeln, nicht in Pixeleinheiten angegeben.

Hat ein VBX-Standardereignis ein Shifttastenargument, dann hat dieses folgende Bitwerte:

Tabelle 2.33: Shifttasten-bitwerte

Taste	Bitwert
Shift	0x1
Ctrl	0x2
Alt	0x4 (wird in Verbindung mit einer Menüauswahl verwendet)

Hat ein VBX-Standardereignis ein Button-Tastenargument, dann hat dieses folgende Bitwerte:

Tabelle 2.34: Maustastenargumente

Taste	Bitwert
Links	0x1
Rechts	0x2
Mitte	0x4

Das folgende Beispiel zeigt, wie Sie diese Shift-Tastenargumente verwenden könnten. Wenn etwa das VBX-Element eine bestimmte Handlung durchführen soll, während die Maus in Bewegung und die Umschalttaste gedrückt ist, könnten Sie eine Funktion dieser Art schreiben:

```
void EvMouseMove(VBXEVENT FAR* event)
{
   short X = VBX_EVENTARGNUM(event, short, 0);
   short Y = VBX_EVENTARGNUM(event, short, 1);
   short Shift = VBX_EVENTARGNUM(event, short, 2);
   short Button = VBX_EVENTARGNUM(event, short, 3);
   if (shift & 0x2)
      MessageBox ("Die Steuertaste ist gedrückt.");
}
```

Borland C++ verwendet Pixel, um die X- und Y-Koordinatenargumente von VBX-Standardereignissen auszudrücken. Hierin unterscheidet es sich von Visual Basic, das Koordinaten in Pixeleinheiten (Twips; 1/20 eines Punkts oder 1/1440 eines Inch) ausdrückt. Benutzerdefinierte Ereignisse sind normalerweise in Twips ausgedrückt. Sie können die folgenden Funktionen verwenden, um Pixel in Pixeleinheiten umzuwandeln und umgekehrt.

Funktion	Bedeutung
VBXPix2TwpX()	Wandelt ein X-Argument von Pixeln in Twips um
VBXPix2TwpY()	Wandelt ein Y-Argument von Pixeln in Twips um
VBXTwp2PixX()	Wandelt ein X-Argument von Twips in Pixel um
VBXTwp2PixY()	Wandelt ein Y-Argument von Twips in Pixel um

Geschützte Elementfunktionen

EvVbxDispatch

```
LRESULT EvVbxDispatch(WPARAM wp, LPARAM lp);
```

Nachdem *TVbxEventHandler* eine WM_VBXFIREEVENT-Botschaft vom übergeordneten Fenster erhalten hat, ruft es *EvVbxDispatch* auf, das die Botschaft an die passende Ereignisbehandlungsfunktion sendet und der VBXEVENT-Struktur einen Zeiger übergibt.

Antworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_MESSAGE(WM_VBXFIREEVENT, EvVbxDispatch)	EvVbxDispatch

Klasse TView — docview.h

TView, virtuell abgeleitet von *TEventHandler* und *TStreamableBase*, ist die einem Dokument präsentierte Schnittstelle, mit der es Zugriff auf seine Client-Views erhält. Die Views rufen dann Dokumentfunktionen zur Anforderung von Ein- und Ausgabestreams auf. Die Views sind die Besitzer der Streams und für ihre Zuordnung und Löschung zuständig.

Anstelle einer Instanz von *TView* erzeugen Sie eine abgeleitete Klasse mit Zugriffsmöglichkeit auf die virtuellen Funktionen von *TView*. Die abgeleitete Klasse muß eine Möglichkeit zur Ermittlung des zugehörigen Fensters (über *GetWindow*), zur Beschreibung des View (über *GetViewName*) und zur Anzeige einer Auswahl von Daten (*GetSelection*) haben. Das View muß außerdem fähig sein, die Ansicht zu einem späteren Zeitpunkt wiederherzustellen (*SetSelection*) und den Titel des Dokuments in seinem Fenster anzuzeigen (*SetDocTitle*).

TView verwendet verschiedene Ereignisbehandlungsfunktionen, um Views abzufragen, festzulegen und zu schließen. Um beispielsweise Views zur Ermittlung desjenigen mit dem Fokus abzufragen, wird, mit der Übergabe eines Handle an das aktuelle Fenster, die Funktion *vnIsWindow* verwendet.

View-Klassen können verschiedene Formen haben. Eine View-Klasse kann z. B. (durch Erben) selbst ein Fenster sein, ein Fenster (als eingebettetes Objekt) enthalten, auf ein Fenster verweisen oder in einem Fensterobjekt enthalten sein. Eine View-Klasse muß nicht einmal ein Fenster besitzen, wie etwa im Falle eines Mail für Sprachbotschaften oder eines

Formatkonverters. Einige Fernviews (z. B. die von OLE 2.0- oder DDE-Servern angezeigten) könnten keine lokalen Fenster besitzen.

Öffentliche Datenelemente

Property enum

```
enum {
  PrevProperty = 0,
  ViewClass,
  ViewName,
  NextProperty,
};
```

Diese für *TView* definierten Eigenschafswerte sind in von *TView* abgeleiteten Klassen verfügbar. PrevProperty und NextProperty sind Begrenzungszeichen für die Attributslisten jedes Dokuments.

Tag

```
LPVOID Tag;
```

Tag speichert einen Zeiger auf die durch die Anwendung definierten Daten. Typischerweise können Sie *Tag* zur Einrichtung eines Zeigers auf die mit Ihrer Anwendung verknüpften Datenstruktur verwenden. Während der Erstellung setzt *TView Tag* auf 0 und greift im folgenden nicht wieder darauf zu.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TView(TDocument& doc);
```

Erstellt ein *TView*-Objekt des mit der Ansicht verknüpften Dokuments. Setzt *ViewId* auf *NextViewId*. Ruft *TDocument::AttachView* auf, um die Ansicht mit dem zugehörigen Dokument zu verbinden.

Destruktor

```
virtual ~TView();
```

Befreit ein *TView*-Objekt und ruft *DetachView* auf, um die Ansicht von dem zugehörigen Dokument zu trennen.

Öffentliche Elementfunktionen

FindProperty

```
virtual int FindProperty(const char far* name);
```

FindProperty ermittelt aus dem gegebenen Eigenschaftsnamen (*name*) den Eigenschaftsindex. Liefert 0 zurück, wenn der Name nicht gefunden wurde.

Siehe auch *pfxxxx property access*-Konstanten

GetDocument

```
TDocument& GetDocument();
```

Liefert eine Referenz zum Dokument des View zurück.

GetNextViewId

```
inline static unsigned GetNextViewId();
```

Liefert die nächste zuzuweisende View-ID zurück.

GetProperty

```
virtual int GetProperty(int index, void far* dest, int textlen=0);
```

Liefert die Gesamtzahl der Eigenschaften zurück, wobei *index* für den Eigenschaftsindex steht, *dest* die Eigenschaftsdaten enthält und *textlen* die Größe des Eigenschaftsarrays ist. Wenn *textlen* 0 ist, werden die Eigenschaftsdaten als binäre, ansonsten als Textdaten zurückgeliefert.

Siehe auch *pfxxxx property access*-Konstanten, *TView::SetProperty*

GetViewId

```
unsigned GetViewId;
```

Liefert die eindeutige ID für dieses View zurück.

GetViewMenu

```
inline TMenuDescr* ViewMenu;
```

Liefert den Menü-Deskriptor für dieses View zurück. Dies kann jedes vorhandene *TMenuDescr*-Objekt sein. Gibt es keinen Deskriptor, ist *ViewMenu* 0.

GetViewName

```
inline virtual LPCSTR GetViewName()=0;
```

Rein virtuelle Funktion, die 0 zurückliefert. Sie müssen, um den Namen der Klasse zurückzuliefern, diese Funktion in Ihrer abgeleiteten Klasse überschreiben.

Siehe auch *TEditView::StaticName*, *TEditView::GetViewName*

GetWindow

```
inline virtual TWindow* GetWindow()
```

GetWindow liefert die mit dem View verknüpfte *TWindow*-Instanz zurück oder 0, wenn es kein View gibt.

Siehe auch *TeditView::GetWindow*

IsOK

```
inline bool IsOK();
```

Liefert Nichtnull zurück, wenn das View erfolgreich erstellt wurde.

Siehe auch *TView::NotOK*

PropertyCount

```
inline virtual int PropertyCount();
```

Speichert die Gesamtzahl der Eigenschaften für das *TDocument*-Objekt. Liefert *NextProperty* - 1 zurück.

Siehe auch *pfxxxx property access*-Konstanten

PropertyFlags

```
virtual int PropertyFlags(int index);
```

Liefert über den gegebenen Index (*index*) einer Eigenschaft, deren Attribute ermittelt werden sollen, die Attribute der spezifizierten Eigenschaft zurück.

Siehe auch *pfxxxx property access*-Konstanten, *TView::FindProperty*, *TView::PropertyName*

PropertyName

```
virtual const char* PropertyName(int index);
```

Liefert mit Hilfe des gegebenen Indexwerts den Textnamen der Eigenschaft zurück.

Siehe auch *pfxxxx property access*-Konstanten, *TView::FindProperty*

SetDocTitle

```
inline virtual bool SetDocTitle(LPCSTR docname, int index)
```

Speichert den Titel des Dokuments.

Siehe auch *TWindow::SetDocTitle*

SetProperty

```
inline virtual bool SetProperty(int index, const void far* src);
```

Setzt den Wert der Eigenschaft mit Hilfe des gegebenen Eigenschaftsindexes und *src*, dem Datentyp (binär oder Text), auf den die Eigenschaft gesetzt werden muß.

Siehe auch *pfxxxx property access*-Konstanten, *TView::GetProperty*

SetViewMenu

```
inline TMenuDescr* ViewMenu;
```

Setzt den Menüdeskriptor für diese Ansicht, die jedes existierende *TMenuDescr*-Objekt sein kann. Gibt es keinen Menüdeskriptor, so ist *ViewMenu* 0.

Siehe auch *TView::GetViewMenu*

Geschützte Datenelemente

Doc

```
TDocument* Doc;
```

Enthält das aktuelle Dokument.

Geschützte Elementfunktionen

NotOK

```
inline void NotOK();
```

Setzt die Ansicht in einen ungültigen Zustand und bewirkt so, daß *IsOK* 0 zurückliefert.

Siehe auch *TView::IsOK*

Aufzählung TWidthHeight — layoutco.h

```
enum TWidthHeight;
```

TWidthHeigt, eine von der Struktur *TLayoutConstraint* verwendete Aufzählung, listet die Werte auf, die Breite (*ImWidth*) und Höhe (*ImHeight*) des Fensters bestimmen.

Siehe auch *TLayoutConstraint* struct

Struktur TWindowAttr window.h

Eine *TWindowAttr*-Struktur wird dazu verwendet, die während eines Erstellungsvorgangs gesetzten Attribute eines *TWindow* zu speichern. Ihr Programm steuert die Erstellung eines Fensters durch Übergabe dieser Werte an eine der Erzeugungsroutinen von *TWindow*. Wenn das Fenster in einen Stream gelesen wird, werden diese Attribute auch für die Neuerstellung verwendet.

Öffentliche Datenelemente

AccelTable

```
TResID AccelTable;
```

AccelTable speichert die Ressourcen-ID für die Tastenkürzeltabelle des Fensters.

Siehe auch *TApplication::HAccTable*, *TWindow::LoadAcceleratorTable*

ExStyle

```
uint32 ExStyle;
```

ExStyle enthält die Werte für erweiterte Eigenschaften Ihres Fensters. Dies kann jede der Windows-API-Konstanten für erweiterten Stil (WS_EX_DLGMODALFRAME, WS_EX_NOPARENTNOTIFY, WS_EX_TOPMOST, WS_EX_SHADOW) sein. In der Windows-API-Online-Hilfe finden Sie weitere Informationen über diese Konstanten.

Id

```
int Id;
```

Id enthält den Bezeichner des untergeordneten Fensters; im Falle eines Dialogfeldes ist *Id* die Ressourcen-ID. Wenn Win32 definiert ist, wird *Id* auf *GetWindowLong*, ansonsten auf *GetWindowWord* gesetzt.

Menu

```
TResID Menu;
```

Menu enthält den Ressourcenbezeichner, der den Namen des Menüs des Fensters repräsentiert, wenn eines mit dem Fenster verknüpft ist.

Param

```
char far* Param;
```

Param enthält einen Wert, der an Windows übergeben wird, wenn das Fenster erzeugt wird. Dieser Wert bestimmt einen Datenblock, der dann in den mit WM_CREATE verbundenen Botschaftsbeantwortungsfunktionen zur Verfügung steht. *Param* wird von *TMDIClient* verwendet und kann bei der Umwandlung von Nicht-ObjectWindows-Code hilfreich sein.

Siehe auch *TWindow::EvCreate*

Style

```
uint32 Style;
```

Style enthält die Werte, die Stil, Form und Größe Ihres Fensters bestimmen. Obwohl *TWindow Attr.Style* auf WS_CHILD und WS_VISIBLE setzt, können Sie auch andere Windows-API-Eigenschaftskonstanten verwen-

den. Eine Beschreibung der *WS_xxxx*-Eigenschaftskonstanten für die Fenstererstellung finden Sie in der Windows-API-Online-Hilfe.

X, Y, W, H

```
int X, Y, W, H;
```

X und *Y* enthalten die Bildschirmkoordinaten der oberen linken Ecke des Fensters. *W* und *H* enthalten die Werte für Breite und Höhe des Fensters.

Klasse TWindow window.h

TWindow, abgeleitet von *TEventHandler* und *TStreamableBase*, stellt fensterspezifische Verhaltensweisen zur Verfügung und kapselt viele der Windows-API-Funktionen. *TWindow*-Funktionen liefern Spezifikationen für die Fenstererstellung, einschließlich der Registrierung und der Attribute.

TWindow ist ein generisches Fenster, dessen Größe neu bestimmt und das bewegt werden kann. Sie können auch eine Instanz von *TWindow* erstellen, obwohl es normalerweise eher als Grundlage für spezialisierte Fensterklassen verwendet wird. Im allgemeinen müssen Sie, um ein *TWindow*-Objekt mit einem Fensterelement zu verknüpfen, folgende Schritte vollziehen:

1. Eine Instanz eines *TWindow* erstellen.
2. *Create* oder *Execute* aufrufen, welches das Oberflächenelement erzeugt und dann *SetupWindow* aufruft. Dieses ruft seinerseits das Basis-*SetupWindow* für normale Verarbeitung auf, das daraufhin
   - die Erstellung des *HWindow* und möglicher untergeordenter *HWindows*
   - sowie den Aufruf von *TransferData* zur Einrichtung des Datentransfers zwischen über- und untergeordneten Fenstern in Gang setzt.
3. Wählen Sie, um das Oberflächenelement zu löschen, abhängig von Ihrer Anwendung eine der folgenden Aktionen:
   - Rufen Sie *Destroy* auf, um das Oberflächenelement bedingungslos zu löschen.
   - Rufen Sie *CloseWindow* auf, das seinerseits *CanClose* aufruft, um zu überprüfen, ob die Löschung des Oberflächenelements in Ordnung ist.
4. Es gibt zwei Möglichkeiten, das Oberflächenobjekt zu löschen:
   - Rufen Sie, wenn das Objekt mit **new** erneuert wurde, *Delete* auf.

- Wurde das Objekt nicht erneuert, löscht der Compiler es automatisch.

Der Löschprozeß von ObjectWindows ist zweiteilig: (1) Rufen Sie *Destroy* auf, um das Oberflächenelement zu entfernen und (2) löschen Sie dann das C++ Objekt. Es ist jedoch auch zulässig, *Destroy* für das Oberflächenelement aufzurufen, ohne das C++ Objekt zu löschen und zu einem späteren Zeitpunkt *Create* aufzurufen, um das Fenster neu zu erzeugen. Da es ebenfalls zulässig ist, ein C++ Fensterobjekt auf dem Stack oder als aggregiertes Element zu erzeugen, kann die Funktion *Destroy* nicht davon ausgehen, daß sie das C++ Objekt löschen sollte.

Die vom Anwender erzeugte Ereignisbehandlungsroutine WM_CLOSE *EvClose* bewirkt ebenfalls durch Übergabe des Zeigers **this** an die Anwendung, daß ein C++ Objekt gelöscht wird. Das C++ Objekt wird automatisch gelöscht, da das Ereignis *EvClose* häufig als Antwort auf eine Aktion des Anwenders auftritt. Außerdem ist dies die geeignetste Stelle für den Löschvorgang. Die Anwendung löscht später, wenn dies gefahrlos durchgeführt werden kann, den Fensterzeiger. Da der Stack häufig Selektoren enthält, die sich auf die Adressen von Objekten beziehen, die während des Löschvorgangs ungültig werden könnten, kann der Zeiger **this** nicht gefahrlos gelöscht werden, während noch Ereignisse verarbeitet werden. Werden die Adressen ungültig, kann dies zu Problemen führen, wenn sie erneut vom Stack geladen werden.

TWindow dient vielen Klassen als Basisklasse, einschließlich *TFrameWindow*, *TControl*, *TDialog* und *TMDIChild*. Das ObjectWindows-Hierarchiediagramm zeigt die von *TWindow* abgeleiteten Klassen.

Öffentliche Datenelemente

Attr

```
TWindowAttr Attr;
```

Attr besitzt eine *TWindowAttr*-Struktur, welche die Attribute für die Erzeugung eines Fensters enthält. Diese Attribute schließen Fensterstil, erweiterte Eigenschaften, Position, Größe, Menü-ID, ID des untergeordneten Fensters und ID der Menükürzeltabelle ein.

Siehe auch *TWindow::TWindow*, *TWindow::Create*, *TwindowAttr*-Struktur

DefaultProc

```
WNDPROC DefaultProc;
```

Speichert die Adresse der Standardprozedur für Fenster. *DefWindowProc* ruft *DefaultProc* auf, um Windows-Botschaften zu verarbeiten, die von der Anwendung nicht behandelt werden.

Siehe auch *TWindow::DefWindowProc*

HWindow

```
HWND HWindow;
```

Enthält das Handle zum verknüpften MS-Windows-Fenster, auf das Sie zugreifen müssen, wenn Sie Windows API-Funktionen direkt aufrufen.

Parent

```
TWindow* parent;
```

Zeigt auf das Oberflächenobjekt, das dem vorliegenden Oberflächenobjekt als übergeordnetes Fenster dient.

Scroller

```
TScroller* Scroller;
```

Zeigt auf das Oberflächenobjekt, das dem vorliegenden Oberflächenobjekt als horizontale oder vertikale Bildlaufleiste dient.

Status

```
TStatus Status;
```

Status wird verwendet, um einen Fehler bei der Initialisierung eines Oberflächenobjekts zu signalisieren. Das Setzen von *Status* auf einen Wert ungleich Null bewirkt, daß eine *TXIncompatibility*-Exception ausgelöst wird. Von *TWindow* abgeleitete Klassen versuchen nicht, ein Oberflächenobjekt mit einem Element zu verknüpfen, dessen vorherige Initialisierung fehlgeschlagen ist. *Status* ist eingeschlossen, um rückgerichtete Kompatibilität zu früheren Versionen von ObjectWindows zu gewährleisten.

Title

```
char far* Title;
```

Title zeigt auf die Titelleiste eines Fensters. Gibt es ein gültiges HWindow, so liefert *Title* dieselben Informationen wie *::GetWindowText*, wenn Sie zur Erstellung *TWindow::SetCaption* verwenden.

Siehe auch *TDialog::SetCaption*, *TDialog::SetupWindow*, *TWindow::GetWindowTextTitle*, *TWindow::SetCaption*

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TWindow(HWND hWnd, TModule* module = 0);
```

TWindow hat zwei Konstruktoren. Der erste erzeugt ein als Alias für ein Nicht-ObjectWindows-Fenster verwendetes *TWindow* und setzt wfAlias. Anders als der zweite *TWindow* Konstruktor, führt dieser das »Thunking« und die Gewinnung von HWND-Informationen durch, anstatt zu warten, bis die Funktion *Create* das Oberflächenelement erzeugt hat.

Beide Formen: Die folgenden Abschnitte beschreiben Verfahren, die beide Konstruktoren aufweisen. *inst* spezifiziert die zur *TWindows*-Instanz gehörende Anwendung oder das DLL-Modul. ObjectWindows benötigt den korrekten Wert von *module*, um die benötigten Ressourcen zu finden. Wenn *module* 0 ist, setzt *TWindow* sein Modul nach folgenden Regeln:

- Hat das Fenster ein übergeordnetes Fenster, wird das *inst* dieses übergeordneten Fensters kopiert.

- Wird der *TWindow*-Konstruktor von einer Anwendung aufgerufen, wird das Modul auf diese Anwendung gesetzt.
- Wenn der *TWindow*-Konstruktor von einer DLL aufgerufen wird, die dynamisch mit der ObjectWindows-DLL verkettet ist, und wenn die gegenwärtig laufende Anwendung auf die gleiche Weise verkettet ist, wird die Bibliothek auf die gegenwärtig laufende Anwendung gesetzt.
- Wenn der *TWindow*-Konstruktor von einer DLL aufgerufen wird, die statisch mit der ObjectWindows-Bibliothek verkettet ist oder wenn die aufrufende DLL dynamisch mit der ObjectWindows-DLL verkettet ist, die laufende Anwendung jedoch nicht, wird zum Setzen des Moduls keine Voreinstellung verwendet. Stattdessen wird eine *TXInvalidModule*-Exception ausgelöst und das Objekt nicht erzeugt.

Siehe auch *TWindow::Init*, *TWindowFlag*-Aufzählung

Konstruktor

```
TWindow(TWindow* parent, const char far* title, TModule* module);
```

Fügt der untergeordneten Liste von *parent*, wenn diese einen Wert ungleich Null besitzt, *this* hinzu und ruft *EnableAutoCreate* auf, so daß *this* zusammen mit *parent* erstellt und angezeigt wird. Setzt ferner den Titel des Fensters und initialisiert die Attribute für die Erstellung des Fensters.

Eine Beschreibung der beiden Konstruktoren gemeinsamen Verfahren finden Sie beim vorhergehenden Konstruktor.

Destruktor

```
virtual ~TWindow();
```

Löscht durch Aufruf von *Destroy* ein noch verknüpftes Oberflächenelement. Die Fensterobjekte in der untergeordneten Liste werden gelöscht und darauf *this* aus der untergeordneten Liste des übergeordneten Fensters entfernt. Ist *Scroller* ungleich Null, wird er gelöscht. Setzt, wenn einer existiert, den Cursor sowie die Objektinstanz (Thunk) frei.

Siehe auch *TWindowFlag*-Aufzählungstyp, *TWindow::EnableAutoCreate*

Öffentliche Elementfunktionen

AdjustWindowRect

```
static void AdjustWindowRect(TRect& rect, uint32 style, bool menu);
```

AdjustWindowRect berechnet die Größe des Fensterrechtecks gemäß der angegebenen Größe des Client-Rechtecks. *rect* referiert auf die Struktur, die die Koordinaten des Client-Rechtecks enthält, *style* spezifiziert den Fensterstil. *menu* ist **true**, wenn das Fenster ein Menü besitzt.

AdjustWindowRectEx

```
static void AdjustWindowRectEx(TRect&, uint style, bool menu,
                               uint32 exStyle);
```

AdjustWindowRectEx berechnet die Größe eines Fensterrechtecks im erweiterten Stil. *TRect* referiert auf die Struktur, welche die Koordinaten des Client-Rechtecks enthält. *style* spezifiziert die Fensterstile des

einzustellenden Fensters und *menu* liefert **true** zurück, wenn das Fenster ein Menü besitzt. *exStyle* bezeichnet den erweiterten Stil, der für das Fenster verwendet werden soll. Erweiterte Stile umfassen die folgenden Stile:

Wert	Bedeutung
WS_EX_ACCEPTFILES	Das Fenster unterstützt Dateioperationen mittels Mausziehen.
WS_EX_DLGMODALFRAME	Das Fenster besitzt eine doppelte Umrandung, die mit einer Titelleiste erzeugt werden kann, wenn das Stil-Flag WS_CAPTION gesetzt ist.
WS_EX_NOPARENTNOTIFY	Das untergeordnete Fenster, das mit diesem Stil erzeugt wird, sendet keine Benachrichtigungsbotschaften an das übergeordnete Fenster, wenn das untergeordnete Fenster erzeugt oder entfernt wird.
WS_EX_TOPMOST	Ein Fenster, dem dieser Stil zugeordnet wird, wird über Fenster gesetzt, die nicht im Vordergrung sind. Es bleibt selbst dann, wenn es deaktiviert wird, über den Fenstern, die nicht im Vordergrund sind.
WS_EX_TRANSPARENT	Ein Fenster, das diesen Stil besitzt, ist durchsichtig. Fenster, die unter diesem transparenten Fenster liegen, werden von ihm nicht verdeckt.

Siehe auch*TWindowsAttr*-Struktur

BringWindowToTop

```
inline void BringWindowToTop();
```

BringWindowToTop bringt ein Pop-Up- oder untergeordnetes Fenster zuoberst eines Stapels sich überlappender Fenster und aktiviert dieses Fenster.

CanClose

```
virtual bool CanClose();
```

Verwenden Sie diese Funktion, um festzustellen, ob Sie ein Fenster schließen können. Liefert **true** zurück, wenn das zugehörige Oberflächenelement geschlossen werden kann. Ruft die *CanClose*-Elementfunktion jedes seiner untergeordneten Fenster auf. Liefert **false** zurück, wenn einer der *CanClose*-Aufrufe **false** zurückliefert.

Sie können im Hauptfenster Ihrer Anwendung *TWindow*s *CanClose* überschreiben und *TWindow::MessageBox* aufrufen, um eine Botschaft YES-NOCANCEL anzuzeigen und den Anwender um die folgende Bestätigung zu bitten:

JA - Daten sichern oder

NEIN - Daten nicht sichern, das Fenster aber schließen oder

ABBRECHEN - Den Schließvorgang abbrechen und in das Editierfenster zurückkehren.

Das folgende Beispiel zeigt eine *CanClose*-Funktion, die den Anwender in einem Meldungsfenster fragt, ob er eine Zeichnung, die sich geändert hat,

speichern möchte. Um Zeit zu sparen, verwendet *CanClose* das Flag *IsDirty* und überprüft, ob die Zeichnung geändert wurde. Ist dies der Fall, erhält der Anwender vor dem Schließen des Fensters von *CanClose* die Abfrage.

```
bool TMyWindow::CanClose()
{
  if (IsDirty)
    switch(MessageBox("Möchten Sie Speichern?", "Die Zeichnung wurde
geändert.",
                      MB_YESNOCANCEL | MB_ICONQUESTION)) {
      case IDCANCEL:
        // Wird "ABBRECHEN" gewählt, wird der Schließvorgang
abgebrochen — Rückgabewert false.
        return false;

      case IDYES:
        // Wird "JA" gewählt, wird die Zeichnung gespeichert .
        CmFileSave();
    }
  return true;
}
```

CheckDlgButton

```
void CheckDlgButton(int buttonId, uint check);
```

Plaziert eine Checkmarke an dem in *buttonId* spezifizierten Schalter (oder entfernt sie von dort). Ist *check* ungleich Null, wird die Checkmarke neben dem Schalter plaziert, ist es 0, wird die Checkmarke entfernt. Für Schalter mit drei möglichen Zuständen kann *check* 0 (hell), 1 (markiert) oder 2 (grau) sein.

CheckRadioButton

```
void CheckRadioButton(int firstButtonId, int lastButtonId,
                      int checkButtonId);
```

Überprüft das durch *checkButtonId* angegebene Schaltfeld und entfernt die Checkmarke von den anderen Schaltfeldern der Gruppe. *firstButtonId* und *lastButtonId* bezeichnen das erste bzw. letzte Schaltfeld in der Gruppe.

ChildBroadcastMessage

```
ChildBroadcastMessage(uint msg, WPARAM wParam, LPARAM lParam);
```

Sendet eine Botschaft an alle unmittelbar untergeordneten Fenster, die *SendMessage* verwenden.

Siehe auch *TWindow::SendMessage*

ChildWindowFromPoint

```
HWND ChildWindowFromPoint(const TPoint&) const;
```

Ermittelt, welches der untergeordneten Fenster den in *TPoint* spezifizierten Punkt enthält. Liefert ein Handle auf das den Punkt enthaltende Fenster zurück, oder 0, wenn der Punkt außerhalb des übergeordneten Fensters liegt.

Siehe auch *TWindow::WindowFromPoint*

ChildWithId

```
TWindow* ChildWithId(int id);
```

Liefert einen Zeiger auf das die angegebene ID enthaltende Fenster in der Liste der untergeordneten Fenster zurück. Liefert 0 zurück, wenn keines der untergeordneten Fenster die angegebene *id* aufweist.

ClearFlag

```
void ClearFlag(TWindowFlag mask);
```

Hebt die Flags von *TWindow- wfXxxx*-Konstanten im Element *Flags* auf (z.B. wfAlias, wfTransfer usw.).

Siehe auch *TWindowFlag*-Aufzählungstyp

ClientToScreen

```
void ClientToScreen(TPoint& point) const;
```

Konvertiert die in *TPoint* spezifizierten Client-Koordinaten in Bildschirmkoordinaten für das neue Fenster.

CloseWindow

```
virtual void CloseWindow(int retval = 0);
```

Stellt vor dem tatsächlichen Schließen des Fensters fest, ob das Fenster geschlossen werden kann. Wenn *this* das Hauptfenster der Anwendung ist, wird *GetApplication->CanClose* aufgerufen. Andernfalls wird *this->CanClose* aufgerufen, um festzustellen, ob das Fenster geschlossen werden kann. *CloseWindow* ruft *Destroy* auf, um das HWND zu entfernen, nachdem festgestellt wurde, daß das Fenster geschlossen werden kann.

Siehe auch *TApplication::CanClose*, *TWindow::CanClose*

CmExit

```
void CmExit();
```

CmExit wird als Reaktion auf einen Menübefehl aufgerufen, der eine ID von CM_EXIT erhalten hat. Wenn *this* das Hauptfenster ist, ruft *CmExit CloseWindow* auf.

Create

```
virtual bool Create();
```

Erzeugt das Windows-Oberflächenelement, das mit dem vorliegenden ObjectWindows-Oberflächenelement verknüpft werden soll.

CreateCaret

```
void CreateCaret(HBITMAP hBitmap);
```

Erstellt einen neuen Cursor für das System. *HBITMAP* spezifiziert die Form des Cursor nach einem Bitmap.

CreateCaret

```
void CreateCaret(bool isGray, int width, int height);
```

Erstellt einen neuen Cursor für das System in der spezifizierten Form, Bitmap-Schattierung, Breite und Höhe. Wenn *width* oder *height* 0 ist, wird die entsprechende vom System definierte Rahmengröße verwendet.

CreateChildren

```
bool CreateChildren();
```

Erstellt diejenigen untergeordneten Fenster in der Liste der untergeordneten Fenster, deren Autocreate-Flags (mit der wfAutoCreate-Maske) gesetzt sind.

Siehe auch *TWindow::EnableAutoCreate*, *TWindow::DisableAutoCreate*, *TWindowFlag*-Aufzählungstyp

DefaultProcessing

```
LRESULT DefaultProcessing();
```

DefaultProcessing dient als Standard-Verarbeitungsfunktion für allgemeine Zwecke, die eine Vielzahl von Botschaften behandelt. Bevor jedoch *DefaultProcessing* aufgerufen wird, schließt ein Fenster die folgende Ereigniskette ab:

- Wenn das Fenster bereits erstellt wurde, wird *SubclassWindow* verwendet, um *StdWndProc* anstelle der aktuellen Prozedur des Fensters einzurichten.
- Wurde das Fenster noch nicht erstellt, wird *InitWndProc* als Fensterprozedur in der Klasse eingerichtet. Wenn das Fenster dann seine erste Botschaft empfängt, ruft *InitWndProc GetModule* auf, um das Instanz-Thunk (vom Konstruktor durch Aufruf von *CreateInstanceThunk* erzeugt) zu erhalten. *InitWndProc* stellt darauf die Fähigkeit zum Empfang von Botschaften der Fensterprozedur auf *StdWndProc* um.

Ist dies geschehen, antwortet *StdWndProc* durch Aufruf der virtuellen *WindowProc*-Prozedur des Fensters auf eingehende Botschaften. Object Windows verwendet die spezielle registrierte Botschaft *GetWindowPtrMsgId*, um den this-Zeiger eines HWND zu erhalten. *StdWndProc* antwortet durch Rückgabe des this-Zeigers, den es vom Thunk empfangen hat.

Ist die eingehende Botschaft kein Befehl, durchsucht *WindowProc* sofort die Antworttabelle des Fensters nach einem passenden Eintrag. Wenn die eingehende Botschaft ein Befehl ist, ruft *WindowProc EvCommandEnable* oder *EvCommand* auf. *EvCommandEnable* und *EvCommand* beginnen, nach einem passenden Eintrag in der Antworttabelle des zentralen Fensters zu suchen. Wurde ein Eintrag gefunden, so wird die entsprechende Funktion gesendet; andernfalls ruft ObjectWindows *DefaultProcessing* auf, um den Rücklauf über die Kette übergeordneter Fenster abzuschließen, indem solange nach einer Übereinstimmung gesucht wird, bis das empfangende Fenster (d.h. das Fenster, das die Botschaft ursprünglich erhalten hat) erreicht ist. An dieser Stelle vollzieht sich eine der folgenden Handlungen:

- Ist das *MainWindow* der Anwendung erreicht und wurde noch immer keine Übereinstimmung gefunden, so durchsucht das Fenster die Antworttabelle der Anwendung.
- Gibt es keine Übereinstimmung und liegt ein Befehl vor, wird *DefWindowProc* aufgerufen.
- Liegt eine *CommandEnable*-Botschaft vor, werden keine weiteren Schritte unternommen.

- Liegt kein Befehl vor und existiert ein Antworttabelleneintrag für das Fenster, so entsendet *WindowProc* die entsprechende *EvXxxx*-Funktion, um die Botschaft zu bearbeiten.
- Ist das *MainWindow* der Anwendung erreicht und die Botschaft für die Anwendung entworfen, wird sie an die Anwendung weitergereicht.
- In allen anderen Fällen ruft das Fenster *DefWindowProc* auf.

Siehe auch *TWindow::DefWindowProc*

DefWindowProc

```
virtual LRESULT DefWindowProc(uint msg, WPARAM wParam, LPARAM lParam);
```

Ruft die subklassierte Fensterprozedur *DefaultProc* auf, um Windows-Verarbeitung voreinzustellen und übergibt die eingehende Windows-Botschaft. Für gewöhnlich muß diese Funktion nicht direkt aufgerufen werden. Klassen wie *TMDIFrame* und die Klasse *DefWindowProc* von *TMDIChild* setzen diese Funktion außer Kraft, um spezialisierte Standardverarbeitung durchzuführen.

Siehe auch *TWindow::DefaultProc*, *TWindow::WindowProc*, *TMDIFrame::DefWindowProc*

Destroy

```
virtual void Destroy();
```

Zunächst ruft *Destroy* für jedes Fenster in der Liste untergeordneter Fenster *EnableAutoCreate* auf, um sicherzustellen, daß Fenster in der Liste untergeordneter Fenster neu erstellt werden, wenn *this* neu erstellt wird. Dann wird das zugehörige Oberflächenelement gelöscht.

Wenn eine abgeleitete Fensterklasse erwartet, direkt gelöscht zu werden, sollte der erste Schritt des Löschvorgangs in einem Aufruf von *Destroy* bestehen, so daß während der Löschsequenz jede beliebige virtuelle Funktion und jede Art von Ereignishandler aufgerufen werden können.

Siehe auch *TWindow::EnableAutoCreate*

DestroyCaret

```
static void DestroyCaret();
```

DestroyCaret überprüft zunächst die Eigentümerverhältnisse des Caret. Ist ein Fenster des aktuellen Task Eigentümer des Fensters, löscht *DestroyCaret* das Caret und entfernt es vom Bildschirm.

Siehe auch *TWindow::CreateCaret*

DisableAutoCreate

```
void DisableAutoCreate();
```

Deaktiviert die Eigenschaft, die es dem Oberflächenelement eines verknüpften untergeordneten Fensters erlaubt, zusammen mit seinem übergeordneten Fenster erstellt und angezeigt zu werden. Sie können *DisableAutoCreate* aufrufen, wenn Sie Popup-Fenster und Dialogelemente erstellen und zu einem späteren Zeitpunkt als ihre übergeordneten Fenster anzeigen lassen möchten.

Siehe auch *TWindow::EnableAutoCreate*

DisableTransfer

```
void DisableTransfer();
```

Deaktiviert (für das Oberflächenobjekt) den Transfermechanismus, der es statischen Daten erlaubt, in einen und aus einem Transferpuffer übertragen zu werden.

Siehe auch *TWindowFlag*-Aufzählungstyp

Dispatch

```
virtual LRESULT Dispatch(TEventInfo& info, WPARAM wp, LPARAM lp = 0);
```

Extrahiert eine *TWindow*-Botschaft und teilt diese zu. Der Parameter *info* ist die Ereignisbehandlungsfunktion. Die Parameter *wp* und *lp* sind die Botschaftsparameter, die von der Zuteilungsfunktion extrahiert werden.

DragAcceptFiles

```
void DragAcceptFiles(bool accept);
```

Wenn ein Fenster abgelegte Dateien verarbeiten kann, setzt *DragAcceptFiles accept* auf **true**.

DrawMenuBar

```
void DrawMenuBar();
```

DrawMenuBar zeichnet die Menüleiste neu. Diese Funktion sollte aufgerufen werden, um das Menü neu zu zeichnen, wenn es nach Erstellung des Fensters geändert wurde.

EnableAutoCreate

```
void EnableAutoCreate();
```

Stellt sicher, daß das Oberflächenelement eines verknüpften untergeordneten Fensters zusammen mit seinem übergeordneten Fenster erstellt und angezeigt wird. Per Voreinstellung ist diese Eigenschaft für Fenster und Dialogelemente aktiviert, nicht jedoch für Dialogfelder.

Siehe auch *TWindow::DisableAutoCreate*

EnableScrollBar

```
void EnableScrollBar(uint sbFlags = SB_BOTH, uint arrowFlags =
ESB_ENABLE_BOTH);
```

Aktiviert oder deaktiviert einen oder beide der Bildlaufleistenpfeile auf den Bildlaufleisten, die mit diesem Fenster assoziiert sind. *sbFlags* gibt die Art der Bildlaufleiste an und kann eine der Bildlaufleisten-Konstanten sein (SB_CTL, SB_HORZ, SB_VERT oder SB_BOTH). Per Voreinstellung sind die Pfeile sowohl der horizontalen wie auch der vertikalen Bildlaufleisten entweder aktiviert oder deaktiviert. *arrowFlags* gibt an, ob die Bildlaufleisten aktiviert oder deaktiviert sind und kann eine der Bildlaufleisten-Aktivieren-Konstanten (ESB_ENABLE_BOTH, ESB_DISABLE_LTUP, ESB_DISABLE_RTDN, ESB_DISABLE_BOTH) sein. Per Voreinstellung sind die Pfeile sowohl der horizontalen wie auch der vertikalen Bildlaufleisten aktiviert.

Siehe auch *SB_xxx*-Bildlaufleisten-Konstanten

EnableTransfer

```
void EnableTransfer();
```

Aktiviert den Transfermechanismus, der es ermöglicht, statische Daten zwischen dem Fenster und einem Transferpuffer zu übertragen.

EnableWindow

```
virtual bool EnableWindow(bool enable);
```

Ermöglicht es dem gegebenen Fenster, Eingaben von der Tastatur oder der Maus zu empfangen. Wenn *enable* **true** ist, kann das Fenster Eingaben empfangen. Verwenden Sie die Funktion *IsWindowEnabled*, um festzustellen, ob das Fenster aktiviert wurde.

Siehe auch *TWindow::IsWindowEnabled*

EnumProps

```
int EnumProps(PROPENUMPROC proc);
```

Zählt die Einträge in der Eigenschaftsliste des aktuellen Fensters auf und übergibt sie nacheinander in die durch *proc* angegebene Callback-Funktion. Dieses Verfahren wird fortgesetzt, bis jeder Eintrag aufgezählt wurde oder bis *proc* Null zurückgibt. *proc* enthält die Adresse der Callback-Funktion.

EvChildInvalid

```
void EvChildInvalid(HWND hwnd);
```

Antwortet auf eine vom Editierfeld eines untergeordneten Fensters abgeschickte WM_CHILDINVALID-Botschaft. Zeigt an, daß der Inhalt des untergeordneten Fensters ungültig ist.

EvCommand

```
virtual LRESULT EvCommand(uint id, HWND hWndCtl, uint notifyCode);
```

WindowProc ruft *EvCommand* zur Bearbeitung von WM_COMMAND-Botschaften auf. *id* ist die Kennung des Menüeintrags oder -elements. *hWndCtl* speichert einen das Dialogelement, welches die Botschaft sendet, repräsentierenden Wert. Geht die Botschaft nicht von einem Dialogelement aus, ist dieser Wert 0. *notifyCode* enthält einen die Benachrichtigungsbotschaft des Dialogelements repräsentierenden Wert. Geht die Botschaft von einem Kurzbefehl aus, ist *notifyCode* 1; geht sie von einem Menü aus, ist *notifyCode* 0.

Siehe auch *TWindow::DefaultProcessing*

EvCommandEnable

```
virtual void EvCommand(TCommandEnabler& ce);
```

EvCommand, das von *WindowProc* zur Bearbeitung von WM_COMMAND_ENABLE-Botschaften aufgerufen wird, ruft die *CmXxxxx*-Befehls-Behandlungsfunktion oder *DefaultProcessing* auf, um die eingehende Botschaft zu bearbeiten.

Siehe auch *TWindow::DefaultProcessing, TCommandEnabler*

EvSysCommand

```
void EvSysCommand (uint cmdType, TPoint& point);
```

Antwortet, wenn der Benutzer einen Befehl aus dem Systemmenü oder das Vergrößerungs- bzw. Verkleinerungssymbol auswählt. Anwendungen, die das Systemmenü verändern, müssen *EvSysCommand*-Botschaften

verarbeiten. Jede von der Anwendung nicht behandelte *EvSysCommand*-Botschaft muß an *DefWindowProc* übergeben werden. Der Parameter *cmdType* kann einer der folgenden Systembefehle sein:

Konstante	Bedeutung
SC_CLOSE	Das Fenster schließen.
SC_HOTKEY	Das spezifizierte Fenster aktivieren.
SC_HSCROLL	Einen horizontalen Bildlauf durchführen.
SC_KEYMENU	Ein Menü durch einen Tastendruck abrufen.
SC_MAXIMIZE (oder SC_ZOOM)	Das Fenster vergrößern.
SC_MINIMIZE (oder SC_ICON)	Das Fenster verkleinern.
SC_MOUSEMENU	Ein Menü durch einen Mausklick abrufen.
SC_NEXTWINDOW	In das nächste Fenster springen.
SC_PREVWINDOW	In das vorherige Fenster springen.
SC_SCREENSAVE	Den spezifizierten Bildschirmspeichervorgang durchführen.
SC_SIZE	Die Größe des Fensters bestimmen.
SC_TASKLIST	Den Windows Task-Manager aktivieren.
SC_VSCROLL	Einen vertikalen Bildlauf durchführen.

Im folgenden Beispiel verarbeitet *EvSysCommand* entweder Systembotschaften oder ruft *DefaultProcessing* auf:

```
void MyWindow::EvSysCommand(uint cmdType, TPoint&)
{
  switch (cmdType & 0xFFF0) {
    case SC_MOUSEMENU:
    case SC_KEYMENU:
      break;
    default:
      DefaultProcessing();
  }
}
```

Siehe auch *TWindow::DefaultProcessing*

FirstThat

```
TWindow* FirstThat(TCondFunc test, void* paramList = 0);
```

```
TWindow* FirstThat(TCondMemFunc test, void* paramList = 0);
```

Es gibt zwei *FirstThat*-Funktionen, die beide einen Zeiger auf eine Iteratorfunktion übergeben. Das eine *FirstThat* zeigt auf eine Nichtelement-Funktion als seinem ersten Parameter; das andere *FirstThat* (siehe den folgenden Eintrag) zeigt stattdessen auf eine Elementfunktion.

Beide *FirstThat*-Funktionen iterieren über die untergeordnete Liste, rufen hierbei eine Boolesche *test*-Funktion auf und übergeben der Reihe nach (zusammen mit *paramList*) jedes untergeordnete Fenster als Argument. Liefert ein Aufruf von *test* **true** zurück, wird die Iteration angehalten und *FirstThat* liefert das Objekt eines untergeordneten Fensters zurück, das *test* angegeben wurde. Andernfalls liefert *FirstThat* 0 zurück.

Im folgenden Beispiel ruft *GetFirstChecked FirstThat* auf, um einen Zeiger (*p*) auf das erste Markierungsfeld der überprüften Liste zu erhalten:

```
bool IsThisBoxChecked(TWindow* p, void*) {
   return ((TCheckBox*)p)GetCheck() == BF_CHECKED;
}
TCheckBox* TMyWindow::GetFirstChecked() {
   return FirstThat(IsThisBoxChecked);
}
```

Siehe auch *TCondFunc*-Typdefinition

FlashWindow

```
bool FlashWindow(bool invert);
```

FlashWindow schaltet das Fenster von aktiv auf inaktiv oder umgekehrt. Ist *invert* ungleich Null, blinkt das Fenster auf. Wenn *invert* 0 ist, wird das Fenster in seinen ursprünglichen Zustand - aktiv oder inaktiv - zurückversetzt.

ForEach

```
void ForEach(TActionFunc action, void* paramList = 0);
```

Es gibt zwei *ForEach*-Funktionen. Die eine hat als ersten Parameter eine Nichtelementfunktion, die andere (siehe den folgenden Eintrag) stattdessen eine Elementfunktion. Beide Versionen von *ForEach* iterieren über die untergeordnete Liste, rufen hierbei eine Nichtelementfunktion auf, die als zu vollführende *action* angegeben wurde und liefern der Reihe nach (zusammen mit *paramList*) jedes untergeordnete Fenster als Argument zurück.

Im folgenden Beispiel ruft *CheckAllBoxes ForEach* auf und überprüft alle Markierungsfelder in der untergeordneten Liste:

```
void CheckTheBox(TWindow* p, void*) {
   ((TCheckBox*)p) Check();
}
void CheckAllBoxes() {
   ForEach(CheckTheBox);
}
```

ForEach

```
void ForEach(TActionMemFunc action, void* paramList = 0);
```

Beachten Sie die vorhergehende Beschreibung von *ForEach*. Der Unterschied zwischen den beiden *ForEach*-Elementen besteht darin, daß das erste eine Nichtelementfunktion als Parameter annimmt, das zweite eine Elementfunktion.

Siehe auch *TActionFunc*-Typdefinition, *TActionMemFunc*-Typdefinition

ForwardMessage

```
LRESULT ForwardMessage(bool send = true)
```

Befördert die aktuelle Botschaft des Fensters. Ruft *SendMessage* auf, wenn *send* **true** ist; andernfalls wird *PostMessage* aufgerufen.

Siehe auch *TWindow::PostMessage*

ForwardMessage

```
LRESULT ForwardMessage(HWND hWnd, bool send = true);
```

Befördert die aktuelle Botschaft des Fensters zu einem anderen HWND. Ruft *SendMessage* auf, wenn *send* **true** ist; andernfalls wird *PostMessage* aufgerufen.

Siehe auch *TWindow::PostMessage*

GetActiveWindow

```
static HWND GetActiveWindow();
```

GetActiveWindow ermittelt das Handle des aktiven Fensters. Liefert 0 zurück, wenn kein aktives Fenster mit dem aufrufenden Thread verknüpft ist.

GetApplication

```
TApplication* GetApplication();
```

Erhält einen Zeiger auf das mit this verknüpfte *TApplication*-Objekt. Verwenden Sie *GetApplication*, um auf Daten und Funktionen im *TApplication*-Objekt zugreifen zu können.

GetCapture

```
static HWND GetCapture();
```

Gibt das Handle des Fensters zurück, das die Mausbotschaften empfangen hat.

GetCaretBlinkTime

```
static uint GetCaretBlinkTime();
```

GetCaretBlinkTime ermittelt die Blinkfrequenz des Caret in Millisekunden.

Siehe auch *TWindow::SetCaretBlinkTime*

GetCaretPos

```
static void GetCaretPos(TPoint& point);
```

GetCaretPos ermittelt die Position des Caret in den Koordinaten des Client-Fensters. *point* referiert auf die Struktur, die die Client-Koordinaten des Caret empfängt.

Siehe auch *TWindow::SetCaretPos*

GetChildren

```
void GetChildren(ipstream &is);
```

Liest untergeordnete Fenster aus dem angegebenen Stream in eine untergordnete Liste ein. *GetChildren* sollte nur während einer *read*-Operation aufgerufen werden, um untergeordnete Fenster zu lesen, die von *Put Children* in den Stream geschrieben wurden.

Diese Elementfunktion nimmt an, daß die untergeordnete Liste gegenwärtig leer ist.

GetClassLong

```
long GetClassLong(int index) const;
```

GetClassLong ermittelt den 32-Bit-Wert der Windows-Klasse. Bei Mißerfolg wird 0 zurückgegeben. Abhängig vom Wert in *index* kann *GetClassLong* die folgende Information ermitteln:

Wert	Bedeutung
GCL_CBCLSEXTRA	Mit dieser Klasse assoziierte Speichergröße in Byte
GCL_CBWINDEXTRA	Mit dieser Klasse assoziierte zusätzliche Fenster-Speichergröße
GCL_HBRBACKGROUND	Handle des mit dieser Klasse assoziierten Hintergrundpinsels
GCL_HCURSOR	Handle des Cursors
GCL_HICON	Handle des Symbols
GCL_HMODULE	Handle des Moduls, das die Klasse eingetragen hat
GCL_MENUNAME	Adresse des Namens-Strings des Menüs
GCL_STYLE	Die mit einer Fensterklasse assoziierten Stil-Bits
GCL_WNDPROC	Adresse der mit dieser Klasse assoziierten Fensterprozedur

Siehe auch *Twindow::SetClassLong*

GetClassWord

```
uint16 GetClassWord(int index) const;
```

GetClassWord ermittelt einen 16-Bit-Wert, der Informationen zu der Klasse oder dem Stil des Fensters enthält. Bei Mißerfolg wird 0 zurückgegeben. Abhängig vom Wert von *index* kann *GetClassWord* die folgenden Informationen ermitteln:

Wert	Bedeutung
GCW_CBCLSEXTRA	Bytezahl der zusätzlichen Klasseninformation
GCW_CBWINDEXTRA	Bytezahl der zusätzlichen Fensterinformation
GCW_HBRBACKGROUND	Handle des Hintergrundpinsels
GCW_HCURSOR	Handle des Cursors
GCW_HICON	Handle des Symbols
GCW_HMODULE	Handle des Moduls
GCW_STYLE	Die mit einer Fensterklasse assoziierten Stil-Bits

Siehe auch *TWindow::SetClassWord*

GetClientRect

```
TRect GetClientRect() const;
```

GetClientRect ermittelt die Koordinaten des Client-Bereichs des Fensters (dies ist der Bereich eines Fensters, in dem gezeichnet werden kann).

GetClientRect

```
void GetClientRect(TRect& rect) const;
```

Ermittelt die Koordinaten des Client-Bereichs des Fensters und kopiert sie dann in das Objekt, auf das sich *TRect* bezieht.

Siehe auch *::GetClientRect*

GetCursorPos

```
static void GetCursorPos(TPoint& pos);
```

GetCursorPos ermittelt die aktuelle Position des Cursors (in den Koordinaten des Bildschirms des Client-Fensters) und kopiert die Werte in die Struktur, auf die *TPoint* zeigt.

GetDesktopWindow

```
static HWND GetDesktopWindow();
```

GetDesktopWindow liefert ein Handle zum Desktop-Fenster zurück.

GetDlgCtrlID

```
int GetDlgCtrlID() const;
```

GetDlgCtrlID liefert die Kennung des Dialogelements zurück.

GetDlgItem

```
HWND GetDlgItem(int childId) const;
```

GetDlgItem ermittelt das Handle eines durch *childId* spezifizierten Dialogelements.

Siehe auch *TWindow::GetDlgItemInt*

GetDlgItemInt

```
uint GetDlgItemInt(int childId, bool *translated = 0,bool isSigned =
                   true) const;
```

GetDlgItemInt ermittelt den Text eines durch *childId* spezifizierten Dialogelements. *translated* zeigt auf die Variable, die den übersetzten Wert empfängt. *isSigned* zeigt an, daß der ermittelte Wert signed (der Vorgabewert) ist.

Siehe auch *TWindow::GetDlgItemInt*

GetDlgItemText

```
uint GetDlgItemText(int childId, char far* text, int max) const;
```

GetDlgItemText ermittelt den Text eines durch *childId* spezifizierten Dialogelements. *text* zeigt auf den Textpuffer, der den Text empfangen soll. *max* gibt die maximale Länge des Titels an, der abgeschnitten wird, wenn er diese Länge überschreitet.

Siehe auch *TWindow::SetDlgItemText*

GetFirstChild

```
TWindow* GetFirstChild();
```

Liefert einen Zeiger auf das erste untergeordnete Fenster in der Liste untergeordneter Fenster des Oberflächenobjekts zurück.

Siehe auch *TWindowAttr*-Struktur

GetFocus

```
static HWND GetFocus();
```

Liefert ein Handle für das Fenster, das den Fokus besitzt. Verwenden Sie die Funktion *SetFocus*, um den Tastatur-Fokus auf dieses Fenster zu setzen.

Siehe auch *TWindow::SetFocus*

GetHWndState

```
void GetHWndState();
```

Kopiert den Stil, die Koordinaten und die Ressourcen-Kennung (nicht jedoch den Titel) aus dem existierenden HWnd in die *TWindow*-Elemente von ObjectWindows.

GetId

```
int GetId();
```

Liefert *Attr.Id* zurück, die ID, die dazu verwendet wird, das Fenster in der Liste untergeordneter Fenster eines spezifizierten übergeordneten Fensters zu finden.

Siehe auch *TWindowAttr*-Struktur

GetModule

```
TModule* GetModule() const;
```

Liefert einen Zeiger auf das Modulobjekt zurück.

GetLastActivePopup

```
HWND GetLastActivePopup() const;
```

Liefert das letzte aktive Popup-Fenster in der Liste zurück.

GetLastChild

```
TWindow* GetLastChild();
```

Liefert einen Zeiger auf das letzte untergeordnete Fenster in der Liste untergeordneter Fenster des Oberflächenobjekts zurück.

GetMenu

```
HMENU GetMenu() const;
```

GetMenu ermittelt das Handle zum Menü des angegebenen Fensters. Der Rückgabewert ist 0, wenn das Fenster kein Menü hat.

Siehe auch *TWindow::SetMenu*

GetNextDlgGroupItem

```
HWND GetNextDlgGroupItem(HWND hWndCtrl, bool previous = false) const;
```

GetNextDlgGroupItem liefert entweder das nächste oder das vorhergehende Dialogelement des Dialogfelds zurück. *hWndCtrl* identifiziert das Dialogelement im Dialogfeld, in dem die Suche beginnt. *GetNextDlgGroupItem* sucht nach dem nächstfolgenden Element, wenn *previous* 0 ist, nach dem vorhergenden, wenn *previous* ungleich Null ist.

GetNextDlgTabItem

```
HWND GetNextDlgTabItem(HWND HWndCtrl, bool previous = false) const;
```

GetNextDlgTabItem liefert ein Handle zum ersten Dialogelement zurück, das den Benutzer die Tabulatortaste drücken läßt, um auf das nächste Element zu springen (d.h. das Handle, mit dem der WS_TABSTOP-Stil verknüpft ist). *hWndCtrl* identifiziert das Dialogelement im Dialogfeld, in dem die Suche beginnt. *GetNextDlgGroupItem* sucht nach dem nächstfolgenden Element, wenn *previous* 0 ist, nach dem vorhergehenden, wenn *previous* ungleich Null ist.

GetNextWindow

```
HWND GetNextWindow(uint dirFlag) const;
```

GetNextWindow findet das mit dem folgenden oder dem vorhergehenden Fenster in der Fensterverwaltungsliste verknüpfte Handle. *dirFlag* spezifiziert die Suchrichtung. Unter der API von WIN32 liefert *GetNextWindow* entweder das Handle des folgenden oder des vorhergehenden Fensters zurück. Läuft die Anwendung nicht unter Win32, liefert *GetNextWindow* das Handle des nächstfolgenden Fensters zurück.

GetWindowPtr

```
TWindow GetWindowPtr(HWND hWnd);
```

Diese Version von *GetWindowPtr* ruft *GetWindowPtr* von *TApplication* in der Anwendung, die mit diesem Fenster verknüpft ist, auf. Ist das Hand-

le zu einem Fenster (*hWnd*) gegeben, liefert *GetWindowPtr* den mit dem Fenster verknüpften *TWindow*-Zeiger zurück.

Siehe auch *TWindow::GetWindowPtr*

GetParent

```
HWND GetParent() const;
```

GetParent ermittelt das Handle zum übergeordneten Fenster. Existiert ein solches nicht, wird 0 zurückgeliefert.

Siehe auch *TWindow::SetParent*

GetProp

```
HANDLE GetProp(uint16 atom) const;
```

GetProp liefert ein Handle zur Attributsliste des spezifizierten Fensters zurück. *atom* enthält einen Wert, der den Zeichenstring identifiziert, dessen Handle gefunden werden soll. Null wird zurückgegeben, wenn der angegebene String in der Eigenschaftsliste für dieses Fenster nicht gefunden wird.

GetProp

```
HANDLE GetProp(const char far* string) const;
```

GetProp liefert ein Handle zur Attributsliste des spezifizierten Fensters zurück. Anders als in der vorhergehenden *GetProp*-Funktion zeigt *string* auf den String, dessen Handle ermittelt werden soll. Null wird zurückgegeben, wenn der angegebene String in der Eigenschaftsliste für dieses Fenster nicht gefunden wird.

Siehe auch *TWindow::SetProp*

GetScrollPos

```
int GetScrollPos(int bar) const;
```

GetScrollPos liefert die Position des Positionszeigers auf der Bildlaufleiste zurück. Die zurückgelieferte Position ist relativ zum Bildlaufbereich. Wenn *bar* SB_CTL ist, wird die Position eines Elements in der Bildlaufleiste zurückgegeben; wenn *bar* SB_HORZ ist, wird die Position einer horizontalen Bildlaufleiste zurückgegeben und wenn *bar* SB_VERT ist, wird die Position einer vertikalen Bildlaufleiste zurückgegeben.

Siehe auch *TWindow::GetScrollPos*

GetScrollRange

```
void GetScrollRange(int bar, int &minPos, int &maxPos) const;
```

GetScrollRange liefert die kleinste und die größte Position auf der Bildlaufleiste zurück. Wenn *bar* SB_CTL ist, wird die Position eines Elements in der Bildlaufleiste zurückgegeben; wenn *bar* SB_HORZ ist, wird die Position einer horizontalen Bildlaufleiste zurückgegeben und wenn *bar* SB_VERT ist, wird die Position einer vertikalen Bildlaufleiste zurückgegeben. *minPos* und *maxPos* spezifizieren die untere bzw. obere Grenze des Positionszeigerbereichs. Gibt es keine Bildlaufleisten-Elemente oder entsprechen sie nicht dem Standard, sind *minPos* und *maxPos* Null.

Siehe auch *TWindow::SetScrollRange*, *SB_xxx-Bildlaufleisten-Konstanten*

GetSysModalWindow

```
static HWND GetSysModalWindow();
```

Ermittelt das Handle des modalen Systemfensters.

Siehe auch *TWindow::SetSysModalWindow*

GetSystemMenu

```
HMENU GetSystemMenu(bool revert = false) const;
```

GetSystemMenu auf und liefert ein Handle zum Systemmenü zurück, so daß eine Anwendung Zugriff auf das Systemmenü erhält.

GetThunk

```
WNDPROC GetThunk() const;
```

Liefert den Instanzen-Thunk zurück, ein kleines Stück Programmcode, das mit exportierten Callback-Funktionen verwendet wird (Eine Callback-Funktion ist eine Funktion, die innerhalb eines Programms vorkommt, aber von außerhalb des Programms durch eine Bibliotheksroutine unter Windows aufgerufen wird - beispielsweise eine Dialogfenster-Funktion).

GetTopWindow

```
HWND GetTopWindow()const;
```

GetTopWindow liefert ein Handle zum obersten Fenster zurück, dessen Besitzer das vorliegende übergeordnete Fenster ist. Existieren keine untergeordneten Fenster, so liefert *GetTopWindow* 0 zurück.

GetUpdateRect

```
bool GetUpdateRect(TRect& rect, bool erase = true) const;
```

GetUpdateRect ermittelt die Bildschirmkoordinaten des Rechtecks, das den aktualisierten Bereich des spezifizierten Fensters umschließt. *erase* bestimmt, ob *GetUpdateRect* den Hintergrund des aktualisierten Bereichs löschen soll.

Siehe auch *TWindow::RedrawWindow*

GetUpdateRgn

```
bool GetUpdateRgn(TRegion& region, bool erase = true) const;
```

GetUpdateRgn kopiert den Aktualisierungsbereich eines Fensters in den durch *region* spezifizierten Bereich. Wenn *erase* **true** ist, löscht *GetUpdateRgn* den Hintergrund des aktualisierten Bereichs und zeichnet die Nicht-Client-Bereiche aller untergeordneten Fenster neu. Wenn *erase* **false** ist, wird nicht neu gezeichnet.

Wenn der Aufruf erfolgreich ist, gibt *GetUpdateRgn* einen Wert zurück, der den Typ des Bereichs, der aktualisiert wurde, beinhaltet. Wenn der Bereich keine überlappenden Begrenzungen hat, wird SIMPLEREGION zurückgegeben. COMPLEXREGION wird zurückgeliefert, wenn der Bereich überlappenden Begrenzungen besitzt; NULLREGION, wenn der Bereich leer ist und ERROR, wenn ein Fehler auftritt.

Siehe auch *TWindow::RedrawWindow*

GetWindow

```
HWND GetWindow (uint cmd) const;
```

Gibt das Handle des Fensters zurück, das die angegebene Beziehung zu diesem Fenster besitzt. *cmd* gibt den Typ der Beziehung an.

Siehe auch *TApplication::GetWindowPtr*

GetWindowFont

```
HFONT GetWindowFont();
```

Holt die Schriftart, die das Element zum Zeichnen des Textes verwendet. Der Rückgabewert ist ein Handle der Schriftart, die das Element verwendet.Wird die Standardschriftart des Systems benutzt, gibt *GetWindowFont* Null zurück.

GetWindowLong

```
long GetWindowLong(int index) const;
```

Liefert abhängig von dem in *index* gespeicherten Wert Informationen über das Fenster zurück. Die zurückgelieferten Werte, die Informationen über das Fenster geben, umfassen die folgenden *GWL_Xxxx*-Fensterstil-Konstanten:

Wert	Beschreibung
GWL_EXSTYLE	Der erweiterte Fensterstil.
GWL_STYLE	Der Fensterstil (Position, Gerätekontexterzeugung, Größe usw.)
GWL_WNDPROC -	Die Adresse der Fensterprozedur, die verarbeitet wird.
Für Dialogfenster können zusätzliche Informationben ermittelt werden, beispielsweise:	
DWL_DLGPROC	Die Adresse der Prozedur, die vom Dialogfenster verarbeitet wird.
DWL_MSGRESULT	Der Wert, den eine Botschaft, die von dem Dialogfenster verarbeitet wurde, zurückliefert.
DWL_USER	Zusätzliche Informationen, die zu der Anwendung gehören, wie beispielsweise Zeiger oder Handles, die die Anwendung verwendet.

Siehe auch *TWindow::GetClassLong*

GetWindowPlacement

```
bool GetWindowPlacement(WINDOWPLACEMENT* place) const;
```

GetWindowPlacement ermittelt Informationen zum Anzeige- und Anordnungsformat (Position im normalen-, verkleinerten und vergrößerten Zustand) des Fensters zu ermitteln und speichert diese im Argument *place*.

Siehe auch *TWindow::SetWindowPlacement*, *TWindow::Show*

GetWindowRect

```
void GetWindowRect(TRect& rect) const;
```

Ermittelt die Bildschirmkoordinaten des Fensterrechtecks und kopiert sie in *rect*.

GetWindowRect

```
TRect GetWindowRect() const;
```

Ermittelt die Bildschirmkoordinaten des Fensterrechtecks.

Siehe auch *TWindow::GetClientRect*

GetWindowTask

```
HTASK GetWindowTask() const;
```

Liefert ein Handle auf die Task zurück, die das spezifizierte Fenster erzeugt hat.

GetWindowText

```
int GetWindowText(char far* string, int maxCount) const;
```

GetWindowText kopiert den Titel des Fensters in einen Puffer, auf den *string* zeigt. *maxCount* gibt die Anzahl der in den Puffer zu kopierenden Zeichen an. Eine Zeichenkette, die länger als *maxCount* ist, wird abgeschnitten. *GetWindowText* liefert die Länge des Strings zurück, oder 0, wenn kein Titel existiert.

Siehe auch *TWindow::SetWindowText*, *TWindow::GetWindowTextTitle*, *TWindow::SetCaption*

GetWindowTextLength

```
int GetWindowTextLength() const;
```

GetWindowTextLength liefert die Länge des Titels des spezifizierten Fensters in Zeichen zurück. Wenn das Fenster ein Dialogelement ist, wird die Länge des Textes in diesem Element zurückgeliefert. *GetWindowTextLength* gibt 0 zurück, wenn das Fenster keinen Text enthält.

Siehe auch *TWindow::SetWindowText*

GetWindowTextTitle

```
void GetWindowTextTitle();
```

Aktualisiert das *TWindow* Titel-Datenelement (*Title*) aus der Titelleiste des aktuellen Fensters. *GetWindowTextTitle* wird verwendet, um *Title* stets mit dem aktuellen Fensterzustand synchron zu halten, wenn die Möglichkeit besteht, daß sich der Zustand geändert haben könnte.

Siehe auch *TWindow::SetCaption*, *TWindow::Title*

GetWindowWord

```
uint16 GetWindowWord(int index) const;
```

GetWindowWord ermittelt abhängig vom in *index* gespeicherten Wert Informationen über das Fenster. *GetWindowWord* liefert einen der folgenden Werte zurück, die Informationen über das Fenster geben:

Wert	Bedeutung
GWW_HINSTANCE	Das Instanzen-Handle des Moduls, dem das Fenster gehört.
GWW_HWNDPARENT	Das Handle des übergeorneten Fensters.
GWW_ID	Die ID-Nummer des untergeordneten Fensters.

Siehe auch *TWindow::GetWindowLong*, *TWindow::SetWindowWord*, *TWindow::SetParent*

HandleMessage

```
LRESULT HandleMessage(uint msg, WPARAM wParam = 0, LPARAM lParam = 0);
```

Bearbeitet die an ein Fenster gesendete Botschaft. *HandleMessage* kann direkt aufgerufen werden, um Fensterbotschaften ohne Durchlaufen von *SendMessage* zu bearbeiten.

Siehe auch *TWindow::SendMessage*

HideCaret

```
void HideCaret();
```

HideCaret entfernt das Caret vom spezifizierten Anzeigebildschirm. Obwohl das Caret nicht sichtbar ist, kann es mit *ShowCaret* wieder angezeigt werden.

Siehe auch *TWindow::CreateCaret*, *TWindow::ShowCaret*

HiliteMenuItem

```
bool HiliteMenuItem(HMENU, uint idItem, uint hilite);
```

HiliteMenuItem hebt entweder die oberste Option in einem Menü hervor oder entfernt deren Hervorhebung. *idItem* kennzeichnet die zu verarbeitende Menüoption. *hilite* (das einen Wert enthält, der anzeigt, ob *idItem* hervorgehoben werden oder ob die Hervorhebung entfernt werden soll) kann eine oder mehrere der folgenden Konstanten sein.

Wert	Bedeutung
MF_BYCOMMAND	Der Parameter *idItem* enthält den Bezeichner der Menüeinträge.
MF_BYPOSITION	Der Parameter *idItem* enthält die nullbasierte, relative Position des Menüeintrags.
MF_HILITE	Hebt den Menüeintrag hervor. Wird dieser Wert nicht angegeben, wird die Hervorhebung des Eintrags entfernt.
MF_UNHILITE	Entfernt die Hervorhebung des Menüeintrags.

Wird das Menü auf die angegebene Bedingung gesetzt, liefert *HiliteMenuItem* **true** zurück. Andernfalls wird **false** zurückgegeben.

Siehe auch *TWindow::GetMenu*

HoldFocusHwnd

```
virtual bool HoldFocusHWnd(HWND hWndLose, HWND hWndGain);
```

Antwortet auf die Forderung eines untergeordneten Fensters, sein HWND zu speichern, wenn es den Fokus verliert. Speichert den Fokus des untergeordneten Fensters in *HwndRestoreFocus*.

Siehe auch *TFrameWindow::HoldFocusHwnd*

HWND

```
operator HWND() const;
```

Erlaubt durch implizite Konvertierung von *TWindow* zu HWND, daß ein *TWindow&* als ein HWND in Windows API-Aufrufen verwendet werden kann.

IdleAction

```
virtual bool IdleAction(long idleCount);
```

IdleAction wird aufgerufen, wenn keine Botschaften auf Verarbeitung warten, und führt Idle-Verarbeitung durch, solange **true** zurückgegeben wird. *idleCount* gibt an, wie oft *IdleAction* zwischen Botschaften aufgerufen wurde.

Invalidate

```
void Invalidate(bool erase = true);
```

InvalidateRect macht den gesamten Client-Bereich ungültig (für das Zeichnen markieren). Das Fenster erhält daraufhin eine Botschaft zum Neu-

zeichnen des Fensters. Per Voreinstellung ist der Hintergrund des Client-Bereichs für das Löschen markiert.

Siehe auch *TWindow::Validate*, *::InvalidateRect*

InvalidateRect

```
void InvalidateRect(const TRect&, bool erase = true);
```

InvalidateRect erklärt einen bestimmten Client-Bereich für ungültig. Per Voreinstellung ist der Hintergrund des Client-Bereichs, der ungültig gemacht werden soll, für das Löschen markiert.

Siehe auch *TWindow::ValidateRect*, *TWindow::Invalidate*

InvalidateRgn

```
void InvalidateRgn(HRGN hRgn, bool erase = true);
```

InvalidateRgn macht einen Client-Bereich in einem durch den Parameter *hRgn* spezifizierten Bereich ungültig, wenn die Anwendung eine WM_PAINT-Botschaft empfängt. Von dem Bereich, der ungültig gemacht werden soll, wird angenommen, daß er Client-Koordinaten besitzt. Wenn *hRgn* 0 ist, wird der gesamte Client-Bereich in den Bereich, der aktualisiert werden soll, eingeschlossen. Der Parameter *erase* gibt an, ob der Hintergrund mit dem zu aktualisierenden Bereich gelöscht werden muß, wenn der zu aktualisierende Bereich festgelegt wird. Wenn *erase* **true** ist, wird der Hintergrund gelöscht; wenn *erase* **false** ist, wird der Hintergrund beim Aufruf der Funktion *Paint* nicht gelöscht. Per Voreinstellung ist der Hintergrund dieses Bereichs zum Löschen markiert.

Siehe auch *TWindow::ValidateRgn*, *TWindow::Paint*

IsChild

```
bool IsChild(HWND) const;
```

IsChild ist **true**, wenn das Fenster ein untergeordnetes Fenster oder ein Nachkomme des vorliegenden Fensters ist. Ein Fenster wird als untergeordnetes Fenster betrachtet, wenn es der direkte Nachkomme eines gegebenen übergeordneten Fensters ist und das übergeordnete Fenster sich in einer Kette von Fenstern befindet, die vom ursprünglichen überlappenden oder Popup-Fenster zu dem untergeordneten Fenster führt. *hWnd* gibt das Fenster an, das geprüft werden soll.

IsDlgButtonChecked

```
uint IsDlgButtonChecked(int buttonId) const;
```

IsDlgButtonChecked gibt an, ob der im Integer-Parameter *buttonId* spezifizierte untergeordnete Schalter markiert ist oder ob der Schalter grau dargestellt, markiert oder keins von beidem ist. Ist der Rückgabewert 0, ist der Schalter nicht markiert. Ist der Rückgabewert 1, ist der Schalter markiert. Ist der Rückgabewert 3, ist der Status des Schalters unbestimmt. Diese Funktion sendet eine BM_GETCHECK-Botschaft an das angegebene Schalterelement.

IsFlagSet

```
bool IsFlagSet(TWindowFlag mask);
```

Liefert den Status des Bit-Flag in *Attr.Flags* zurück, dessen *mask* angegeben ist. Liefert **true** zurück, wenn das Bit-Flag gesetzt ist, **false**, wenn es nicht gesetzt ist.

Siehe auch *TWindowAttr*-Struktur

IsIconic

```
bool IsIconic() const;
```

Ist **true**, wenn das Fenster als Symbol dargestellt oder verkleinert ist.

IsWindow

```
bool IsWindow() const;
```

Ist **true**, wenn ein HWND verwendet wird.

IsWindowEnabled

```
bool IsWindowEnabled();
```

Ist **true**, wenn das Fenster aktiviert ist. Verwenden Sie die Funktion *EnableWindow*, um ein Fenster zu aktivieren oder zu deaktivieren.

Siehe auch *TWindow::EnableWindow*

IsWindowVisible

```
bool IsWindowVisible() const;
```

Ist **true**, wenn das Fenster sichtbar ist. Per Voreinstellung richtet der Konstruktor von *TWindow* das Fensterstilattribut (WS_VISIBLE) so ein, daß das Fenster sichtbar ist.

IsZoomed

```
bool IsZoomed() const;
```

Ist **true**, wenn das Fenster vergrößert oder auf Vollbild aufgezogen ist.

KillTimer

```
bool KillTimer(uint timerId);
```

KillTimer auf, die entledigt sich des Timers und entfernt alle WM_TIMER-Botschaften aus der Botschafts-Warteschlange. *timerId* enthält die ID-Kennzahl des zu löschenden Timer-Ereignisses.

Siehe auch *TWindow::SetTimer*

LockWindowUpdate

```
bool LockWindowUpdate();
```

LockWindowUpdate verhindert oder aktiviert das Fensterzeichnen für jeweils ein Fenster. Ist das Fenster gesperrt, wird **true** zurückgegeben. Andernfalls wird **false** zurückgegeben und damit angezeigt, daß ein Fehler aufgetreten ist oder daß ein anderes Fenster bereits gesperrt ist.

Wird in einem gesperrten Fenster oder in gesperrten untergeordneten Fenstern versucht zu zeichnen, wird der Umfang des Versuchs in einem begrenzenden Rechteck gespeichert. Wird die Sperrung des Fensters dann aufgehoben, wird der Bereich innerhalb des Rechtecks für ungültig erklärt. Dies bewirkt, daß eine Zeichnen-Botschaft an dieses Fenster geschickt wird. Der Bereich wird für ungültig erklärt, wenn Zeichenoperationen aufgetreten sind, während das Fenster zur Aktualisierung gesperrt war.

MapWindowPoints

```
void MapWindowPoints(HWND hWndTo, TPoint* points, int count) const;
```

MapWindowPoints verweist eine Menge von Punkten in einem Fenster auf eine verwandte Menge von Punkten in einem anderen Fenster. *hWndTo* spezifiziert das Fenster, in welches die Punkte zu übertragen sind. *points* zeigt auf das die Punkte enthaltende Array. Ist *hWndTo* 0, werden die Punkte in Bildschirmkoordinaten übertragen. *count* spezifiziert die Anzahl der *points*-Strukturen in dem Array.

MessageBox

```
int MessageBox(const char far* text,const char far* caption = 0, uint
type = MB_OK);
```

Erstellt und zeigt ein Meldungsfenster an, das eine Meldung (*text*), einen Titel (*caption*) sowie Symbole oder Aktionsschalter (*type*) enthält. Ist *caption* 0, wird der voreingestellte Titel angezeigt. Obwohl die Voreinstellung für *type* nur aus einem Aktionsschalter besteht, kann der Parameter eine Kombination aus Windows-API-*MB_Xxxx*-Konstanten enthalten. Diese Funktion liefert eine der folgenden Konstanten zurück:

Wet	Beschreibung
IDABORT	Der Anwender hat den Schalter Abbrechen (Abort) gewählt.
IDCANCEL	Der Anwender hat den Schalter Abbrechen (Cancel) ausgewählt.
IDIGNORE	Der Anwender hat den Schalter Ignorieren ausgewählt.
IDNO	Der Anwender hat den Schalter Nein ausgewählt
IDOK	Der Anwender hat den Schalter OK ausgewählt
IDRETRY	Der Anwender hat den Schalter Erneut Versuchen ausgewählt.
IDYES	Der Anwender hat den Schalter Ja ausgewählt.

Wenn BWCC bereits aktiviert ist, wird auch das Meldungsfenster BWCC-fähig sein. Wenn CTRL3D bereits aktiviert ist, wird auch das Meldungsfenster CTRL3D-fähig sein. Wenn weder BWCC noch CTRL3D aktiviert sind, wird das Meldungsfenster als Standard-Windows-Meldungsfenster dargestellt.

Siehe auch *TWindow::PostMessage*, *MB_Xxxx*-Botschafts-Konstanten

MoveWindow

```
void MoveWindow(int x, int y, int w, int h, bool repaint = false);
```

MoveWindow positioniert das spezifizierte Fenster neu. *x* und *y* bezeichnen die neuen Koordinaten für die obere linke Ecke des Fensters, *w* und *h* die neue Breite bzw. Höhe. Wenn *repaint* **false** ist, wird das Fenster nicht neu gezeichnet, nachdem es bewegt wurde.

MoveWindow

```
void MoveWindow(const TRect& rect, bool repaint = false);
```

MoveWindow positioniert das Fenster neu. *rect* bezieht sich auf die linke und die obere Koordinate sowie auf die Breite und Höhe des neuen Bildschirmrechtecks. Wenn *repaint* **false** ist, wird das Fenster nicht neu gezeichnet, nachdem es bewegt wurde.

Siehe auch *TWindow::RedrawWindow*

Next

```
TWindow* Next();
```

Liefert einen Zeiger auf das nächste Fenster in der Liste gleichgeordneter Fenster.

Siehe auch *TWindow::Previous*

NumChildren

```
unsigned NumChildren();
```

Liefert die Anzahl der untergeordneten Fenster von ObjectWindows zurück.

OpenClipboard

```
TClipboard& OpenClipboard();
```

Öffnet die Zwischenablage und hindert andere Anwendungen daran, den Inhalt der Zwischenablage zu verändern. Diese Funktion bleibt ohne Erfolg, wenn ein anderes Fenster die Zwischenablage bereits geöffnet hat.

Paint

```
virtual void Paint(TDC&, bool erase, TRect& rect);
```

Zeichnet den Client-Bereich des Fensters neu (den Bereich, der zum Zeichnen verwendet werden kann). *Paint*, das von Basisklassen aufgerufen wird, wenn diese auf eine WM_PAINT-
Botschaft reagieren, dient als Platzhalter für abgeleitete Typen, die *Paint*-Elementfunktionen definieren. *Paint* wird von *EvPaint* aufgerufen und automatisch von Windows angefordert, um den Inhalt des Fensters neu anzuzeigen. *dc* ist der für Text- und Grafikoutput-Funktionen angegebene Anzeigekontext für den Zeichenvorgang. Die angegebene Referenz zur *rect*-Struktur ist das den für das Zeichnen freigegebenen Bereich begrenzende Rechteck. *erase* zeigt an, ob der Hintergrund gelöscht werden muß.

PostMessage

```
bool PostMessage(uint msg, WPARAM wParam = 0, LPARAM lParam=0) const;
```

PostMessage hinterlegt eine Botschaft (*msg*) in der Botschaftswarteschlange einer Anwendung. Die Funktion kehrt zurück, ohne abzuwarten, bis das entsprechende Fenster die Botschaft verarbeitet hat.

Siehe auch *TWindow::ForwardMessage*, *TWindow::MessageBox*

PerformCreate

```
virtual void PerformCreate(int menuOrId);
```

PerformCreate wird aus *Create* heraus aufgerufen, um den letzten Schritt in der Erzeugung eines MS Windows-Oberflächenelements, das mit einem ObjectWindows-Fenster verknüpft werden soll, durchzuführen. *PerformCreate* kann überschrieben werden, um eine alternative Implementierung der Erzeugung von HWND zu ermöglichen.

Siehe auch *TWindow::Create*

PreProcessMsg

```
virtual bool PreProcessMsg(MSG& msg);
```

PreProcessMsg erlaubt das Pre-Processing von Fenster- und Tastaturbotschaften. Wenn Sie diese Funktion in einer abgeleiteten Klasse außer Kraft setzen, müssen Sie sicherstellen, daß *PreProcessMsg* aufgerufen

wird, da es die Übertragung von Tastenkürzeln steuert. Bei Rückgabe von einem Wert ungleich Null wird die Botschaftsverarbeitung angehalten.

Siehe auch *TApplication::ProcessAppMsg*

Previous

```
TWindow* Previous();
```

Liefert einen Zeiger auf das vorhergehende Fenster in der Liste untergeordneter Fenster des übergeordneten Fensters zurück.

Siehe auch *TWindow::Next*

ReceiveMessage

```
LRESULT ReceiveMessage(uint msg, WPARAM wParam = 0, LPARAM lParam = 0);
```

ReceiveMessage, das von StdWndProc aufgerufen wird, ist die erste nach Empfang einer Botschaft aufgerufene Elementfunktion. Sie ruft ihrerseits *HandleMessage* aus dem try-Block des Exception-Behandlungs-Codes auf. Auf diese Weise können Exceptions abgefangen und aufgeschoben werden, bevor das Dialogelement an den Exception-unsicheren Windows-Code zurückgeliefert wird.

Siehe auch *TWindow::HandleMessage*

RedrawWindow

```
bool RedrawWindow(TRect* update, HRGN hUpdateRgn,
                  uint redrawFlags = RDW_INVALIDATE |
                  RDW_UPDATENOW | RDW_ERASE);
```

RedrawWindow zeichnet das durch *update* angegebene Rechteck an der durch *hUpdateRgn* bezeichneten Adresse neu. *redrawFlags* kann eine Kombination aus einer oder mehreren der folgenden *RDW_Xxxx*-Fenster-Neuzeichnen-Konstanten sein, die ein Fenster validieren oder für ungültig erkären.

Diese Werte erklären ein Fenster für ungültig:

Wert	Beschreibung
RDW_ERASE	Wenn das Fenster neu gezeichnet wird, empfängt es eine WM_ERASEBKGND-Botschaft. Wenn RDW_INVALIDATE nicht ebenfalls angegeben wurde, hat dieses Flag keine Auswirkung.
RDW_FRAME	Jeder Teil des Nicht-Client-Bereichs des Fensters empfängt eine WM_NCPAINT-Botschaft, wenn er sich mit dem zu aktualisierenden Bereich überschneidet.
RDW_INTERNALPAINT	Eine WM_PAINT-Botschaft wird für das Fenster hinterlegt, und zwar unabhängig davon, ob es einen ungültigen Bereich enthält oder nicht.
RDW_INVALIDATE	Erklärt entweder *hUpdateRgn* oder *update* für ungültig. Sollten beide 0 sein, wird das gesamte Fenster ungültig.
Diese Werte validieren ein Fenster:	
RDW_NOERASE	Es wird verhindert, daß das Fenster WM_ERASEBKGND-Botschaften erhält.
RDW_NOFRAME	Es wird verhindert, daß das Fenster WM_NCPAINT-Botschaften erhält. Das Flag RDW_VALIDATE muß mit diesem Flag evenfalls verwendet werden.

RDW_NOINTERNALPAINT	Es wird verhindert, daß das Fenster interne WM_PAINT-Botschaften erhält. Es wird jedoch nicht verhindert, daß das Fenster WM_PAINT-Botschaften von ungültigen Bereichen erhält.
RDW_VALIDATE	Validiert *update* und *hUpdateRgn*. Sind beide 0, wird der gesamte Fensterbereich validiert. Das Flag besitzt keine Auswirkungen auf interne WM_PAINT-Botschaften.
Diese Flags steuern, wann das Fenster neu gezeichnet wird:	
RDW_ERASENOW	Bevor die Funktion zurückkehrt, erhält das angegebene Fenster WM_NCPAINT- und WM_ERASEBKGND-Botschaften.
RDW_UPDATENOW	Bevor die Funktion zurückkehrt, erhält das angegebene Fenster WM_NCPAINT-, WM_ERASEBKGND- sowie WM_PAINT-Botschaften.

Siehe auch *TWindow::GetUpdateRect*

Register

```
virtual bool Register();
```

Registriert die Windows-Registrierklasse von this, wenn this nicht bereits registriert wurde. Ruft *GetClassName* und *GetWindowClass* auf, um den Namen der Windows-Registrierklasse und die Attribute von this zu bestimmen. *Register* liefert **true** zurück, wenn this registriert ist.

Siehe auch *TWindow::GetClassName*, *TWindow::GetWindowClass*, *WNDCLASS*-Struktur

RegisterHotKey

```
bool RegisterHotKey(int idHotKey, uint modifiers, uint virtKey);
```

RegisterHotKey registriert eine Tastenkürzel-ID in der aktuellen Anwendung. *modifiers* kann eine Tastenkombination sein, die gedrückt werden muß, um das spezifizierte Kürzel zu aktivieren, z. B. HOTKEYF_SHIFT, HOTKEYF_CONTROL und HOTKEYF_ALT.

Siehe auch *TWindow::UnRegisterHotKey*

ReleaseCapture

```
static void ReleaseCapture();
```

Gibt den Mausfang dieses Fensters frei.

RemoveProp

```
HANDLE RemoveProp(uint16 atom) const;
```

RemoveProp entfernt die durch *atom* spezifizierte Eigenschaft aus der Eigenschaftsliste der Anwendung. *atom* gibt den String an, der entfernt werden soll. Gibt das Handle des gegebenen Strings oder Null zurück, wenn kein String in der Eigenschaftsliste des Fensters existiert.

RemoveProp

```
HANDLE RemoveProp(const char far* str) const;
```

RemoveProp entfernt die durch *str*, einen nullterminierten String, spezifizierte Eigenschaft aus der Eigenschaftsliste der Anwendung. Gibt das Handle des gegebenen Strings oder Null zurück, wenn kein String in der Eigenschaftsliste des Fensters existiert.

Siehe auch *TWindow::GetProp*

ScreenToClient

```
void ScreenToClient(TPoint& point) const;
```

ScreenToClient verwendet die in *point* spezifizierten Koordinaten zur Berechnung der Koordinaten des Client-Fensters und legt die neuen Koordinaten dann in *point* ab.

ScrollWindow

```
void ScrollWindow(int dx, int dy, const TRect* scroll = 0,
                  const TRect far *clip = 0);
```

ScrollWindow führt einen Bildlauf eines Fensters in vertikaler (*dx*) und horizontaler (*dy*) Richtung durch. *TRect* bezeichnet den Bereich, für den der Bildlauf durchgeführt werden soll. Dieser erfolgt für den gesamten Bereich, wenn *TRect* 0 ist. *clip* spezifiziert das beschnittene zu rollende Rechteck. Nur der in *clip* enthaltene Bereich wird gerollt. Der Bildlauf wird für das gesamte Fenster durchgeführt, wenn *clip* 0 ist.

Siehe auch *TWindow::ScrollWindowEx*

ScrollWindowEx

```
void ScrollWindowEx(int dx, int dy, const TRect *scroll = 0,
                    const TRect far *clip = 0,
                    HRGN hUpdateRgn = 0, TRect far *update = 0,
                    uint flags = 0);
```

ScrollWindowEx führt einen Bildlauf eines Fensters in vertikaler (*dx*) und horizontaler (*dy*) Richtung durchz. *scroll* bezeichnet den Bereich, für den der Bildlauf durchgeführt werden soll. Dieser erfolgt für den gesamten Bereich, wenn *scroll* 0 ist. *clip* spezifiziert das beschnittene zu rollende Rechteck. Nur der in *clip* enthaltene Bereich wird gerollt. Der Bildlauf wird für das gesamte Fenster durchgeführt, wenn *clip* 0 ist. *update* bezeichnet den Bereich, der die Grenzen desjenigen Bereichs empfängt, der als Ergebnis des Bildlaufs ungültig wird. *flags*, das bestimmt, wie die untergeordneten Fenster des Fensters gescrollt werden, kann eine der *SW_Xxxx*-Fensterbildlauf-Konstanten sein:

Wert	Beschreibung
SW_ERASE	Löscht den für ungültig erklärten Bereich, nachdem eine Botschaft zum Löschen des Hintergrunds an das durch den Flag-Wert SW_INVALIDATE angegebene Fenster gesendet wurde.
SW_INVALIDATE	Erklärt den durch den Parameter *hUpdate* angegebenen Bereich für ungültig.
SW_SCROLLCHILDREN	Führt einen Bildlauf für alle untergeordneten Fenster durch, die sich mit Rechteck, aus das der Parameter *scroll* zeigt, überschneiden.

Siehe auch *TWindow::ScrollWindow*, *TWindow::Show*

SendDlgItemMessage

```
LRESULT SendDlgItemMessage(int childId, uint msg,
                           WPARAM wParam = 0,
                           LPARAM lParam = 0) const;
```

SendDlgItemMessage sendet eine Botschaft (*msg*) an das in *childId* spezifizierte Dialogelement.

Siehe auch *TWindow::SendMessage*

SendMessage

```
LRESULT SendMessage(uint msg, WPARAM wParam = 0,
                    LPARAM lParam = 0) const;
```

SendMessage sendet eine Botschaft (*msg*) an ein oder mehrere spezifizierte Fenster. Nach Aufruf der Fensterprozedur wartet *SendMessage*, bevor es zurückkehrt, bis diese die Botschaft verarbeitet hat.

Siehe auch *TWindow::ChildBroadcastMessage*, *TWindow::HandleMessage*, *TWindow::SendDlgItemMessage*

SendNotification

```
void SendNotification(int id, int notifyCode, HWND hCtl,
                      uint msg = WM_COMMAND);
```

Packt eine Befehlsbotschaft (*msg*) neu, so daß ein untergeordnetes Fenster (*hCtl*) eine Botschaft an das ihm übergeordnete senden kann, ohne Rücksicht darauf, ob es sich hier um eine WIN16- oder WIN32-Anwendung handelt.

SetActiveWindow

```
HWND SetActiveWindow();
```

SetActiveWindow aktiviert ein Fenster auf oberster Ebene. Liefert ein Handle für das gegenwärtig aktive Fenster zurück.

Siehe auch *TWindow::GetActiveWindow*

SetBkgndColor

```
void SetBkgndColor(uint32 color);
```

Setzt die Hintergrundfarbe (*color*) für das Fenster. Sie können auch die aktuelle Farbe eines am Bildschirm angezeigten Grafikelements ermitteln.

```
layout SetBkgndColor(GetSysColor(COLOR_APPWORKSPACE));
```

beispielsweise verwendet einen der COLOR-Werte von Windows (in diesem Fall die Farbe von Multidokument-Schnittstellenanwendungen (MDI).

Siehe auch *TWindow::BkgndColor*

SetCaption

```
void SetCaption(const char far* title);
```

Kopiert *title* in einen allozierten String, auf den *Title* zeigt. Setzt den Titel des Oberflächenelements auf *title* und löscht alle vorherigen Titel.

Siehe auch *TWindow::GetWindowTextTitle*, *TWindow::Title*

SetCapture

```
HWND SetCapture();
```

Setzt den Mausfang auf das aktuelle Fenster. Alle Mauseingaben werden an dieses Fenster weitergeleitet.

SetCaretBlinkTime

```
static void SetCaretBlinkTime(uint16 milliSecs);
```

SetCaretBlinkTime setzt die Blinkfrequenz des Caret in Millisekunden.

Siehe auch *TWindow::GetCaretBlinkTime*

SetCaretPos

```
static void SetCaretPos(int x, int y);
```

SetCaretPos setzt die Position des Caret in den Koordinaten des Client-Fensters. *x* und *y* geben die Client-Koordinaten des Caret an.

Siehe auch *TWindow::GetCaretPos*, *TWindow::ShowCaret*

SetCaretPos

```
static void SetCaretPos(const TPoint &pos);
```

SetCaretPos setzt die Position des Caret in den Koordinaten des Client-Fensters. *pos* gibt die Client-Koordinaten des Caret an.

Siehe auch *TWindow::GetCaretPos*, *TWindow::ShowCaret*

SetClassLong

```
long SetClassLong(int index) const;
```

SetClassLong setzt den long-Wert in dem spezifizierten Offset (*index*) zu setzen. In Abhängigkeit vom Wert von *index* setzt *SetClassLong* ein Handle zum Pinsel für den Hintergrund, zum Cursor, zu einem Symbol, einem Modul, einem Menü, einer Fensterfunktion oder klassenfremde Bytes.

Siehe auch *TWindow::GetClassLong*

SetClassWord

```
uint16 SetClassWord(int index, uint16 newWord);
```

SetClassWord setzt den Wortwert auf das spezifizierte Offset (*index*). Abhängig vom Wert von *index* setzt *SetClassLong* die Anzahl der Bytes der Klasseninformationen, zusätzlichen Fensterinformationen oder der Style-Bits. Anders als *SetClassLong* verwendet *SetClassWord* eine der folgenden *GCW_xxxx*-Konstanten:

Wert	Bedeutung
GCW_HBRBACKGROUND	Setzt ein Handle für einen Hintergrundpinsel.
GCW_HCURSOR	Setzt ein Handle für einen Cursor.
GCW_HICON	Setzt ein Handle für eine Symbol.
GCW_STYLE	Setzt ein Stil-Bit für eine Fensterklasse.

Siehe auch *TWindow::GetClassWord*

SetCursor

```
bool SetCursor(TModule* module, TResId resId);
```

Setzt unter Verwendung der gegebenen *module* und *ResId* die Cursorposition für das Fenster. Ist der Parameter *module* 0, kann *CursorResId* eine der IDC_xxxx-Konstanten sein, die verschiedene Cursortypen repräsentieren. Unter dem Datenelement finden Sie eine Liste dieser Cursorwerte. Befindet sich die Maus über dem Client-Bereich, ändert *SetCursor* den Cursor, der angezeigt wird.

Siehe auch *TWindow::GetCursorPos, TWindow::CursorResId*

SetDlgItemInt

```
void SetDlgItemInt(int childId, uint value, bool isSigned = true) const;
```

::SetDlgItem setzt das mit der ID (*childId*) versehene untergeordnete Fenster auf den in *value* spezifizierten Intergerwert. Es handelt sich um einen Signed-Wert, wenn *isSigned* **true** ist.

Siehe auch *TWindow::GetDlgItem*

SetDlgItemText

```
void SetDlgItemText(int childId, const char far* text) const;
```

SetDlgItemText setzt den Text eines mit einer ID versehenen untergeordneten Fensters zu setzen. *text* zeigt auf den Puffer, der den Fenstertitel oder -text enthält, der in den Fenstertitel bzw. -text des untergeordneten Fensters kopiert werden soll.

Siehe auch *TWindow::GetDlgItemText*

SetDocTitle

```
virtual bool SetDocTitle(const char far* docname, int index);
```

Speichert den Titel des Dokuments (*docname*). *index* ist die Nummer der View, die in der Titelleiste des Dokuments angezeigt wird. *SetDocTitle* stellt die Nummer der View in zwei Durchgängen fest: im ersten Durchgang wird überprüft, ob es mehr als eine View gibt. Im zweiten Durchgang wird der View die nächste Nummer zugewiesen, wenn es mehr als eine View gibt. Gibt es jedoch nur eine View, ist *index* 0, und das Dokument zeigt keine View-Nummer an. *index* ist -1, wenn *TDocument* prüft, ob mehr als eine View existiert. In diesem Fall wird nur der Dokumenttitel in der Überschrift angezeigt.

SetDocTitle gibt **true** zurück, wenn es mehr als eine View gibt und wenn *TDocument* die Nummer der View, die in *index* übergeben wurde, anzeigt.

SetFlag

```
void SetFlag(TWindowFlag mask);
```

Wenn **true** angegeben ist, werden die Bits der *Attr.Flags* in *Mask* gesetzt; andernfalls werden die Bits gelöscht. *Mask* kann eine beliebige *wfXxxx*-Konstante oder eine Kombination aus mehreren sein.

Siehe auch *TWindow::IsFlagSet*, *TWindow::Flag* Aufzählungstyp

SetFocus

```
HWND SetFocus();
```

Setzt den Tastaturfokus auf das aktuelle Fenster und aktiviert durch das Senden einer WM_SETFOCUS-Botschaft das Fenster, das den Fokus erhält. Alle folgenden Tastatureingaben werden an dieses Fenster geleitet, und das vorherige Fenster, das den Eingabefokus hatte, verliert ihn. Bei Erfolg wird ein Handle für das Fenster, das den Fokus besitzt, zurückgegeben. Andernfalls wird Null zurückgegeben.

Siehe auch *TWindow::GetFocus*

SetMenu

```
bool SetMenu(HMENU hMenu);
```

Setzt das spezifizierte Fenstermenü auf das durch *hMenu* angegebene Menü. Wenn *hMenu* 0 ist, wird das aktuelle Menü des Fensters entfernt.

SetMenu gibt 0 zurück, wenn das Menü unverändert bleibt. Andernfalls wird ein Wert ungleich Null zurückgegeben.

Siehe auch *TWindow::GetMenu, TMDIFrame::SetMenu*

SetModule

```
void SetModule(TModule* module);
```

Setzt das Standardmodul für dieses Fenster.

Siehe auch *TWindow::GetModule*

SetNext

```
void SetNext(TWindow* next);
```

Setzt das nächste Fenster in der Liste gleichgeordneter Fenster.

SetParent

```
virtual void SetParent(TWindow* newParent);
```

Setzt *Parent* auf das spezifizierte Objekt des übergeordneten Fensters. Entfernt this aus der Liste untergeordneter Fenster des vorhergehenden übergeordneten Fensters, wenn ein solches existiert, und fügt dieses Fenster der Liste untergeordneter Fenster des neuen übergeordneten Fensters hinzu.

Siehe auch *TWindow::GetParent*

SetProp

```
bool SetProp(uint16 atom, HANDLE data) const;
```

SetProp fügt der Eigenschaftsliste des übergeordneten Fensters einen Eintrag hinzu. *atom* enthält einen Wert, der die der Eigenschaftsliste hinzuzufügenden Dateneinträge bezeichnet.

SetProp

```
bool SetProp(const char far* str, HANDLE data) const;
```

SetProp fügt der Eigenschaftsliste des übergeordneten Fensters einen Eintrag hinzu. *str* zeigt auf den String, der die der Eigenschaftsliste hinzuzufügenden Dateneinträge bezeichnet.

Siehe auch *TWindow::GetProp*

SetRedraw

```
void SetRedraw(bool redraw);
```

Sendet eine WM_SETREDRAW-Botschaft an ein Fenster, damit Änderungen neu gezeichnet werden können (redraw = **true**) oder um zu verhindern, daß Änderungen neu gezeichnet werden (redraw = **false**).

SetScrollPos

```
int SetScrollPos(int bar, int pos, bool redraw = true);
```

SetScrollPos setzt die Position des Positionszeigers auf der Bildlaufleiste. *bar* bezeichnet die zurückzuliefernde Position (horizontales, vertikales oder Bildlaufleistenelement) und kann eine der *SB_xxxx*-Bildlaufleisten-Konstanten sein.

Siehe auch *TWindow::GetScrollPos*, *SB_xxxx*-Bildlaufleisten-Konstanten

SetScrollRange

```
void SetScrollRange(int bar, int minPos, int maxPos,
                    bool redraw = true);
```

SetScrollRange setzt die Position des Positionszeigers auf der Bildlaufleiste. *bar* bezeichnet die zu setzende Position (horizontales, vertikales oder Bildlaufleistenelement) und kann eine der *SB_xxxx*-Bildlaufleisten-Konstanten sein. *minPos* und *maxPos* bezeichnen die untere bzw. obere Grenze des Bildlaufbereichs.

Siehe auch *TWindow::GetScrollRange*; *TWindow::SetScrollRange, SB_Xxxx*-Bildlaufleisten-Konstanten

SetSysModalWindow

```
HWND SetSysModalWindow();
```

Macht das bezeichnete Fenster zu einem Modalfenster des Systems.

Siehe auch *TWindow::GetSysModalWindow*

SetTimer

```
uint SetTimer(uint timerId, uint timeout, TIMERPROC proc = 0);
```

SetTimer erstellt ein Timer-Objekt, das mit diesem Fenster verknüpft ist. *timerId* enthält die ID des hinzuzufügenden Zeitereignisses, *timeout* spezifiziert die Zeitspanne in Millisekunden und *proc* bestimmt die Adresse der Funktion, die benachrichtigt werden soll, wenn das zeitgesteuerte Ereignis stattfindet. Wenn *proc* 0 ist, werden die WM_TIMER-Botschaften in der Warteschlange der Anwendung abgesetzt, die *SetTimer* für dieses Fenster aufgerufen hat.

Siehe auch *TWindow::KillTimer*

SetTransferBuffer

```
void SetTransferBuffer(void* transferBuffer);
```

Setzt *TransferBuffer* auf *transferBuffer*.

Siehe auch *TWindow::Transfer*, *TWindow::TransferData*

SetWindowFont

```
void SetWindowFont(HFONT font, bool redraw);
```

Setzt die Schriftart, die ein Element zum Zeichnen von Text verwendet. *font* gibt die Schriftart an, die verwendet wird und ist bei Verwendung der Standard-Systemschriftart Null. Wenn *redraw* **true** ist, zeichnet sich das Element neu, nachdem die Schriftart gesetzt wurde. Wenn *redraw* **false** ist, zeichnet sich das Element nicht neu. Das Beispielprogramm FILEBROW.CPP zeigt, wie die Schriftarten für ein Dateiauswahlfenster gesetzt werden.

SetWindowLong

```
long SetWindowLong(int index, long newLong);
```

SetWindowLong ändert die Informationen über ein Fenster. Abhängig vom Wert von *index* setzt *SetWindowLong* ein Handle auf den Pinsel für den Hintergrund, den Cursor, ein Symbol, ein Modul, ein Menü oder eine Fensterfunktion. Der Fensterstil kann einer der *GWL_xxxx*-Werte sein, Stile repräsentieren. Unter *GetWindowLong* finden Sie eine Beschreibung dieser Werte.

Siehe auch *TWindow::GetWindowLong*, *TWindow::SetWindowWord*

SetWindowPlacement

```
bool SetWindowPlacement(WINDOWPLACEMENT* place);
```

SetWindowPlacement setzt für das Fenster einen Anzeigemodus und eine Bildschirmposition. *place* zeigt auf eine Struktur zum Setzen des Fensters, die angibt, ob das Fenster versteckt, verkleinert, als Symbol angezeigt, auf Vollbild vergrößert, an der vorherigen Position wiederhergestellt, in seiner aktuellen Form aktiviert oder in seiner normalen Position aktiviert und angezeigt werden soll.

Siehe auch *TWindow::GetWindowPlacement*, *TWindow::Show*

SetWindowPos

```
void SetWindowPos(HWND hWndInsertAfter, const TRect &rect,
                  uint flags);
```

SetWindowPos ändert die Größe des Fensters, auf das *rect* zeigt. *flags* enthält eine der *SWP_Xxxx*-Fenster-Setzen-Positionskonstanten, die die Größe und Position des Fensters bezeichnen. Ist *flags* auf SWP_NOZORDER gesetzt, ignoriert *SetWindowPos* den *hWndInsertAfter*-Parameter und behält die gegenwärtige Abfolge der untergeordneten Fenster, Popup-Fenster oder Fenster der obersten Ebene bei.

SetWindowPos

```
void SetWindowPos(HWND hWndInsertAfter, int x, int y, int w,
                  int h, uint flags);
```

SetWindowPos ändert die Größe des Fensters, auf das *x*, *y*, *w* und *h* zeigen. *flags* enthält eine der *SWP_xxxx*-Konstanten, die die Größe und Position des Fensters bezeichnen. Ist *flags* auf SWP_NOZORDER gesetzt, ignoriert *SetWindowPos* den *hWndInsertAfter*-Parameter und behält die gegenwärtige Abfolge der untergeordneten Fenster, Popup-Fenster oder Fenster oberster Ebene bei.

Wert	Beschreibung
SWP_DRAWFRAME	Zeichnet einen Rahmen um das Fenster.
SWP_FRAMECHANGED	Sendet eine Botschaft an das Fenster, um die Größe des Fensters neu zu berechnen. Ist dieses Flag nicht gesetzt, wird eine Botschaft zur Neuberechnung der Größe nur dann gesendet, wenn sich die Größe des Fensters ändert.
SWP_HIDEWINDOW	Versteckt das Fenster.
SWP_NOACTIVATE	Aktiviert das Fenster nicht. Ist dieses Flag nicht gesetzt, wird das Fenster aktiviert und zuoberst auf dem Fenster-Stack plaziert.
SWP_NOCOPYBITS	Entfernt den gesamten Inhalt des Client-Bereichs des Fensters. Ist dieses Flag nicht gesetzt, werden die gültigen Inhalte gespeichert und in das Fenster kopiert, nachdem das Fenster in der Größe verändert oder neu positioniert wurde.
SWP_NOMOVE	Speichert die aktuelle Position des Fensters.
SWP_NOSIZE	Speichert die aktuelle Größe des Fensters.
SWP_NOREDRAW	Zeichnet Änderungen des Fensters nicht neu. Ist dieses Flag gesetzt, findet kein Neuzeichnen irgendeines Fensterbereichs (einschließlich Client, Nicht-Client oder als Ergebnis von Fensterbewegungen sichtbar gewordener Bereiche) statt. Ist dieses Flag

	gesetzt, muß die Anwendung ausdrücklich angeben, daß ein Bereich des Fensters ungültig ist und neu gezeichnet werden muß.
SWP_NOZORDER	Speichert die aktuelle Z-Reihenfolge (Fenster-Stack-Reihenfolge).
SWP_SHOWWINDOW	Zeigt das Fenster an.

Siehe auch *TWindow::GetWindowPlacement*

SetWindowText

```
void SetWindowText(const char far* str);
```

SetWindowText setzt den Titel des Fensters auf den Puffer, auf welchen *str* zeigt. *maxCount* bezeichnet die Anzahl der in den Puffer zu kopierenden Zeichen. Bitte beachten Sie, daß das Element *Title* dieses Fensters hiermit nicht aktualisiert wird. Verwenden Sie *SetCaption*, wenn das Element *Title* des Fensters mit dem Fenstertitel synchronisiert werden soll.

Siehe auch *TWindow::GetWindowText*

SetWindowWord

```
uint16 SetWindowWord(int index, uint16 newWord);
```

SetWindowWord ändert die Informationen über das Fenster. *index* spezifiziert ein Byte-Offset des auf den neuen Wert zu ändernden Wortes (*newWord*).

Siehe auch *TWindow::GetWindowWord*, *TWindow::SetWindowLong*

Show

```
void Show(int showCmd);
```

Nachdem sichergestellt wurde, daß das *TWinddow*-Oberflächenelement ein gültiges Handle besitzt, zeigt *Show* das *TWindow* auf eine durch *showCmd* bestimmte Weise am Bildschirm an. *showCmd* kann eine der *SW_Xxxx*-Zeige-Fenster-Konstanten sein.

ShowCaret

```
void ShowCaret();
```

ShowCaret zeigt den Cursor in der spezifizierten Größe in der aktuellen Position im aktiven Fenster anzuzeigen.

Siehe auch *TWindow::CreateCaret*, *TWindow::HideCaret*

ShowOwnedPopups

```
void ShowOwnedPopups(bool show);
```

Zeigt das Popup-Fenster gemäß dem in *show* angegebenen Wert an.

Siehe auch *TWindow::Show*

ShowScrollBar

```
void ShowScrollBar(int bar, bool show = true);
```

ShowScrollBar zeigt die Bildlaufleiste an oder verbirgt sie. *bar* legt fest, ob die Leiste ein Dialogelement oder Teil des Nicht-Client-Bereichs des Fensters ist. Wenn *bar* Teil des Nicht-Client-Bereichs ist, kann es eine der *SB_xxxx*-Bildlaufleisten-Konstanten (genauer gesagt, SB_BOTH, SB_HORZ oder SB_VERT) sein. Wenn es sich um ein Dialogelement handelt, sollte die Konstante SB_CTRL sein. Die Bildlaufleiste wird angezeigt, wenn *show* **true** ist, ist es **false**, so wird sie verborgen.

Siehe auch *TWindow::GetScrollRange*, *SB_Xxxx*-Bildlaufleisten-Konstanten

ShowWindow

```
virtual bool ShowWindow(int cmdShow);
```

Zeigt das Fenster gemäß dem Wert von *cmdShow* an. Eine Beschreibung der in *show* übergebenen *SW_xxxx*-Konstanten finden Sie unter dem Eintrag *Show*. *Show* ist die bevorzugte Methode für die Anzeige des Fensters.

Siehe auch *TWindow::Show*

ShutDownWindow

```
void ShutDownWindow(int retVal = 0);
```

Diese Inline-Version von *ShutDownWindow* ruft die statische Version von *ShutDownWindow* auf.

ShutDownWindow

```
static void ShutDownWindow(TWindow* win, int retVal = 0);
```

Diese Version von *ShutDownWindow* schließt ohne weitere Bedingungen ein gegebenes Fenster, ruft für das Oberflächenelement *Destroy* auf und löscht dann das Oberflächenelement. Sie können *Destroy* auch direkt aufrufen und anschließend das Oberflächenelement löschen, anstatt *ShutDownWindow* zu verwenden.

SubclassWindowFunction

```
void SubclassWindowFunction();
```

Richtet den Instanzen-Thunk als *WindowProc* ein und speichert die alte Fensterfunktion in *DefaultProc*.

Siehe auch *TWindow::DefaultProc*, *TWindow::DefaultProcessing*

Transfer

```
virtual uint Transfer(void* buffer, TTransferDirection direction);
```

Überträgt Daten in oder aus einem Fenster jeglicher Art mit untergeordneten Fenstern und liefert die Gesamtgröße der übertragenen Daten an den Puffer zurück. *Transfer* ist ein allgemeiner Mechanismus zur Datenübertragung, der mit oder ohne *TransferData()* verwendet werden kann. Die angegebene Richtung *direction* bestimmt, ob Daten in den angegebenen Puffer geschrieben oder aus ihm gelesen werden sollen oder ob nur die Größe der Transferdaten zurückgeliefert werden soll. Sind in irgendwelchen untergeordneten Fenstern keine *wfTransfer*-Flags gesetzt, werden keine Daten zwischen ihnen übertragen. Wenn Daten übertragen werden, ist der Rückgabewert die Größe (in *bTransferData*) der Transferdaten.

TransferData

```
virtual void TransferData(TTransferDirection direction);
```

Ein Fenster ruft *TransferData* normalerweise während der Einrichtung und des Schließens von Fenstern auf und ist auf den Konstruktor angewiesen, um *TransferBuffer* eine Bedeutung zuzuweisen. *TransferData* ruft die Elementfunktion *Transfer* jedes beteiligten untergeordneten Fensters auf, wobei es einen Zeiger sowohl auf *TransferBuffer* als auch auf die in *direction* angegebene Richtung (*tdSetData*, *tdGetData* oder *tdSizeData*) übergibt.

Siehe auch *TWindow::EnableTransfer*, *TWindow::DisableTransfer*, *TWindow::SetupWindow*, *TWindow::TransferBuffer*

UnregisterHotKey

```
bool UnregisterHotKey(int idHotKey);
```

UnregisterHotKey entfernt die ID des Tastenkürzels einer aktuellen Anwendung aus dem Register.

Siehe auch*TWindow::RegisterHotKey*

UpdateWindow

```
void UpdateWindow();
```

UpdateWindow aktualisiert den Client-Bereich des spezifizierten Fensters durch unmittelbares Absenden einer WM_PAINT-Botschaft.

Validate

```
void Validate();
```

Ruft die Funktion *ValidateRect* auf, um den gesamten Client-Bereich (den Bereich, in den gezeichnet werden kann) zu prüfen (d. h. ihn aus dem zu aktualisierenden Bereich zu entfernen).

Siehe auch *TWindow::InValidate*

ValidateRect

```
void ValidateRect(const TRect&);
```

ValidateRect prüft einen bestimmten Teil des durch *rect* bezeichneten Client-Bereichs.

ValidateRgn

```
void ValidateRgn(HRGN hRgn);
```

ValidateRgn prüft den Client-Bereich innerhalb eines Bereichs des aktuellen Fensters. *hRgn* ist ein Handle für den Client-Bereich, der aus dem zu aktualisierenden Bereich entfernt werden soll. Wenn *hRgn* Null ist, wird der gesamte Client-Bereich aus dem zu aktualisierenden Bereich entfernt.

Siehe auch *TWindow::InvalidateRgn*

WindowFromPoint

```
static HWND WindowFromPoint(const TPoint& point) const;
```

Liefert einen Wert zurück, der das Fenster bezeichnet, in dem sich der spezifizierte Punkt (*point*) befindet.

Siehe auch *TWindow::ChildWindowFromPoint*

WindowProc

```
virtual LRESULT WindowProc(uint msg, WPARAM wParam, LPARAM lParam);
```

WindowProc verarbeitet eingehende Botschaften, wobei *EvCommand* zur Bearbeitung von WM_COMMAND-Botschaften, *EvCommandEnable* zur Bearbeitung von WM_COMMAND_ENABLE-Botschaften aufgerufen wird. Außerdem übernimmt *WindowProc* die Zuteilungsfunktionen für alle anderen Botschaften.

Siehe auch *TWindow::DefWindowProc*

WinHelp

```
bool WinHelp(const char far* helpFile, uint command, uint32 data);
```

WinHelp invoziert ein spezifiziertes Hilfe-System. *helpFile* zeigt auf einen Verzeichnispfad und einen die Namen der Hilfedatei enthaltenden String. *command*, das die Art der erwünschten Hilfe bezeichnet, kann eine der Windows-Help_xxxx-Konstanten wie HELP_CONTEXT, HELP_HELPONHELP, HELP_INDEX, HELP_MULTIKEY, HELP_QUIT oder HELP_SETINDEX sein. *data* enthält Schlüsselwörter, die die einzelnen Hilfe-Themen bezeichnen. In der ObjectWindows-Beispieldatei HELP.CPP etwa wird *WinHelp* über die Argumente HELP_CONTEXT und HELP_MENUITEMA abgerufen, wenn [*F1*] gedrückt wird.

```
void TOwlHelpWnd::CmMenuItemA()
{
  if (F1Pressed) {
    WinHelp(HelpFile, HELP_CONTEXT, HELP_MENUITEMA);
    F1Pressed = false;
  } else {
    MessageBox("In Menu Item A command", Title, MB_ICONINFORMATION);
  }
}
```

Sie können ferner Bitmaps in Ihre Hilfedatei aufnehmen, indem Sie auf deren Dateinamen referieren oder sie aus der Zwischenablage kopieren. Weitere Informationen zu Erstellung von Hilfedateien finden Sie im Handbuch zur Online-Hilfe.

Geschützte Datenelemente

BkgndColor

```
uint32 BkgndColor;
```

Speichert die aktuelle Hintergrundfarbe, die durch die Funktion *TWindow::SetBkgndColor* gesetzt wird.

Siehe auch *TWindow::SetBkgndColor*

CursorModule

```
TModule* CursorModule;
```

Enthält die Modul-ID für den angegebenen Cursor. Ein Wert von 0 bedeutet einen Standard-Systemcursor.

Siehe auch *TWindow::CursorResId*

CursorResId

```
TResId CursorResId;
```

Enthält die Cursor-Ressourcen-ID für den spezifizierten Cursor. Wenn das Datenelement *CursorModule* 0 ist, kann *CursorResId* eine der folgenden *IDC_Xxxx*-Konstanten sein, die verschiedene Cursortypen repräsentieren:

Wert	Bedeutung
IDC_ARROW	Gewöhnlicher Pfeilcursor
IDC_CROSS	Kreuzcursor

IDC_IBEAM	Strichcursor
IDC_ICON	Leerer Symbolcursor
IDC_SIZE	Ein kleineres Quadrat in der rechten inneren Ecke eines größeren Quadrats
IDC_SIZENESW	Doppelrichtungscursor mit Pfeilen, die nach Südwesten und Nordosten zeigen
IDC_SIZENS	Doppelrichtungscursor mit Pfeilen, die nach Süden und Norden zeigen
IDC_SIZENWSE	Doppelrichtungscursor mit Pfeilen, die nach Südosten und Nordwesten zeigen
IDC_SIZEWE	Doppelrichtungscursor mit Pfeilen, die nach Osten und Westen zeigen
IDC_UPARROW	Vertikaler Pfeilcursor
IDC_WAIT	Sanduhrcursor

Siehe auch *TWindow::SetCursor*

hAccel

```
HACCEL hAccel;
```

Enthält das Handle zur aktuellen Windows-Tastenkürzeltabelle, die mit diesem Fenster assoziiert ist.

HCursor

```
HCURSOR HCursor;
```

Enthält ein Handle zum Fenstercursor. Der Cursor wird durch *CursorModule* und *CursorResId* ermittelt und durch *SetCursor* gesetzt.

TransferBuffer

```
void* TransferBuffer;
```

TransferBuffer zeigt auf einen Puffer, der bei der Übertragung von Daten in das und aus dem *TWindow* verwendet werden soll. Ein *TWindow* setzt voraus, daß der Puffer Daten enthält, die von den Fenstern in seiner Liste der untergeordneten Fenster verwendet werden. Ist *TransferBuffer* 0, gibt es keine zu übertragenden Daten.

Siehe auch *tdxxxx*-Konstanten

Geschützter Konstruktor

Konstruktor

```
TWindow();
```

Dieser Konstruktor wird mit virtuell abgeleiteten Klassen verwendet. Unmittelbar abgeleitete Klassen müssen *Init* aufrufen, bevor die Erzeugung des Objekts abgeschlossen ist.

Geschützte Elementfunktionen

CleanupWindow

```
virtual void CleanupWindow();
```

CleanupWindow, das unmittelbar vor Ungültigwerden des HWindow aufgerufen wird, verschafft abgeleiteten Klassen die Möglichkeit, mit HWND verbundene Ressourcen aufzuräumen. Diese Funktion stellt das Gegenstück zu *SetupWindow* dar.

Überschreiben Sie diese Funktion in ihren abgeleiteten Klassen zum Aufräumen der Fenster. *CleanupWindow* verschafft abgeleiteten Klassen die Möglichkeit, mit hwnd verbundene Ressourcen aufzuräumen. Abgeleite-

te Klassen sollten die Basisklassen-Version von *CleanupWindow* als letzten Schritt aufrufen, bevor sie zurückkehren. Der folgende Programmcode aus dem Beispielprogramm APPWIN.CPP erläutert diesen Prozeß:

```
//Teilt den Fenstern mit, daß keine weiteren
Ziehen-und-Ablegen-Transaktionen akzeptiert werden und führt weitere
Aufräumarbeiten an den Fenstern durch.
void
TAppWindow::CleanupWindow()
{
  AppLauncherCleanup();
  DragAcceptFiles(false);
  TWindow::CleanupWindow();
}
```

Siehe auch *TWindow::SetupWindow*

DispatchScroll

```
void DispatchScroll(uint scrollCode, uint thumbPos, HWND hWndCtrl);
```

Wird von *EvHScroll* und *EvVScroll* aufgerufen, um Botschaften von Bildlaufleisten zu verteilen.

GetClassName

```
virtual char far* GetClassName();
```

Liefert den Namen der Windows-Registrierklasse zurück. Der Standard-Klassenname wird aus »Window« und dem Namen des Moduls zusammengesetzt. Überschreiben Sie diese Funktion in ihren abgeleiteten Klassen, wenn Sie eine neue Klasse registrieren oder den Namen einer bestehenden Fensterklasse verändern.

Siehe auch *TWindow::GetWindowClass, WNDCLASS*-Struktur

GetWindowClass

```
virtual void GetWindowClass(WNDCLASS& wndClass);
```

Wird von abgeleiteten Klassen neu definiert, um die angegebene MS Windows-
Registrierklassen-Strukur mit Registrierungsattributen zu füllen. Diese Funktion erlaubt, zusammen mit *GetClassName*, die Verwendung von Windows-Klassen für die spezifizierte ObjectWindows-Klasse und deren Derivate. *GetWindowClass* setzt die Felder der übergebenen *WNDCLASS*-Parameter auf die Standard-Eingenschaften, die für ein *TWindow* geeignet sind. Die Felder und ihre Standardeigenschaften für die Klasse sind wie folgt:

Wert	Bedeutung
cbClsExtra	0 (Die Zahl der zusätzlichen Bytes, die nach der Fensterklassen-Struktur reserviert werden müssen). Dieser Wert wird von ObjectWindows nicht verwendet.
cbWndExtra	0 (Die Zahl der zusätzlichen Bytes, die nach der Fenster-Instanz reserviert werden müssen). Dieser Wert wird von ObjectWindows nicht verwendet.
hInstance	Die Instanz der Klasse, in der die Fensterprozedur besteht.

hIcon	0 (Stellt ein Handle zu der Klassen-Ressource zur Verfügung). Per Voreinstellung muß die Anwendung ein Symbol erzeugen, wenn das Fenster der Anwendung zum Symbol verkleinert wird.
hCursor	IDC_ARROW (Stellt ein Handle für eine Cursor-Ressource zur Verfügung).
hbrBackground	COLOR_WINDOW + 1 (Die System-Hintergrundfarbe).
lpszMenuName	0 (Zeigt auf einen String, der den Namen des Menüs der Klasse enthält). Per Voreinstellung besitzen die Fenster in dieser Klasse keine zugewiesenen Menüs.
lpszClassName	Zeigt auf einen String, der den Namen der Fensterklasse enthält.
lpfnWndProc	Die Adresse der Fensterprozedur. Dieser Wert wird von ObjectWindows nicht verwendet.
style	CS_DBLCLKS

Das Feld Stil kann einen oder mehrere der folgenden Werte enthalten:

CS_BYTEALIGNCLIENT	Richtet den Client des Fensters an einer Byte-Begrenzung in x-Richtung aus. Diese Ausrichtung, die die Leistung steigert, bestimmt die Breite und die horizontale Position des Fensters.
CS_BYTEALIGNWINDOW	Richtet das Fenster an einer Byte-Begrenzung in x-Richtung aus. Diese Ausrichtung, die die Leistung steigert, bestimmt die Breite und die horizontale Position des Fensters.
CS_CLASSDC	Weist einen einzelnen Gerätekontext (DC) zu, der von allen Fenstern in der Klasse gemeinsam verwendet wird. Dieser Stil legt fest, wie Multithread-Anwendungen, deren Fenster zur gleichen Klasse gehören, ein DC gemeinsam verwenden.
CS_DBLCLKS	Sendet eine Doppelklick-Mausbotschaft an die Fensterprozedur, wenn in einem zu dieser Klasse gehörenden Fenster ein Doppelklick mit der Maus ausgeführt wird.
CS_GLOBALCLASS	Ermöglicht einer Anwendung, unabhängig vom Instanzen-Parameter eine Klasse zu erzeugen. Eine globale Klasse kann auch durch das Schreiben einer DLL, die die Fensterklasse enthält, erzeugt werden.
CS_HREDRAW	Zeichnet das gesamte Fenster neu, wenn sich ein Fenster als Ergebnis einer Bewegung oder Größenveränderung verändert.
CS_NOCLOSE	Deaktiviert die Schließen-Option auf dem Systemmenü dieses Fensters.
CS_OWNDC	Ermöglicht es jedem Fenster in der Klasse, einen unterschiedlichen Gerätekontext zu verwenden.
CS_PARENTDC	Übergibt den Gerätekontext des übergeordneten Fensters an die untergeordneten Fenster.
CS_SAVEBITS	Speichert den Teilbereich des Bildschirms als Bitmap, wenn der Bildschirm von einem anderen Fenster verdeckt wird. Diese Bitmap wird anschließend dazu benutzt, das Fenster wiederherzustellen, sobald es nicht mehr von einem anderen Fenster verdeckt wird.
CS_VREDRAW	Zeichnet das gesamte Fenster neu, wenn die Höhe des Client-Bereichs verändert wird.

Nachdem die Fensterklassen-Struktur von der Basisklasse mit Standardwerten gefüllt wurde, können Sie diese Funktion überschreiben, um die Werte der Fensterklassen-Struktur zu verändern, beispielsweise, um die Farben des Fensters oder den angezeigten Cursortyp zu modifizieren.

Siehe auch *TWindow::GetClassName, WNDCLASS*-Struktur

Init

```
void Init(TWindow* parent, const char far* title, TModule* module);
```

Ermöglicht weitere Initialisierung nach der Standard-Erzeugung eines Fensters in virtuell abgeleiteten Klassen.

LoadAcceleratorTable

```
void LoadAcceleratorTable();
```

Lädt ein Handle zur in der *TWindowsAttr*-Struktur (*Attr.AccelTable*) spezifizierten Tastenkürzeltabelle des Fensters. Existiert kein Handle, wird die Diagnosemeldung »Unable to load acceleator table« ausgegeben.

Siehe auch *StrukturTWindowsAttr*

RemoveChild

```
void RemoveChild(TWindow* child
```

Entfernt ein untergeordnetes Fenster. Diese Familie von OjectWindows-Funktionen verwendet anstelle der Windows-HWND-Liste eher die ObjectWindows-Objektliste.

Siehe auch *TWindow::GetFirstChild*, *TWindow::GetLastChild*, *TWindow::Next*, *TWindow::Previous*

SetupWindow

```
virtual void SetupWindow();
```

SetupWindow ist die erste virtuelle Funktion, die aufgerufen wird, wenn das HWindow Gültigkeit erlangt. Die Implementierung von *TWindow* richtet zunächst ein Fenster ein und erzeugt dann ein zugehöriges Oberflächenelement. Die Funktion iteriert durch die Liste untergeordneter Fenster und versucht, für jedes untergeordnete Fenster, für das automatische Erstellung aktiviert ist, ein verknüpftes Oberflächenelement zu erzeugen (per Voreinstellung ist Autocreation für Fenster und Dialogelemente aktiviert, für Dialogfelder deaktiviert). *SetupWindow* ruft daraufhin *TransferData* auf.

SetupWindow kann in abgeleiteten Klassen neu definiert werden, um zusätzliche spezielle Initialisierungen durchzuführen. Beachten Sie, daß das HWindow gültig ist, sobald das überschriebene *SetupWindow* aufgerufen wird, und daß die HWindows der untergeordneten Fenster nach Aufruf der *SetupWindow*-Funktion der Basisklasse Gültigkeit erlangen.

Der folgende Programmcode aus dem Beispielprogramm APPWIN.CPP zeigt die Verwendung eines überschriebenen *SetupWindow* für das Einrichten eines Fensters, die Initialisierung der .INI-Einträge und die Mitteilung an Windows, daß Ziehen-und-Ablegen-Transaktionen akzeptiert werden sollen:

```
void TAppWindow::SetupWindow()
{
  TFloatingFrame::SetupWindow();
```

```
    InitEntries();   // Initialisierung .INI-Einträge.
    RestoreFromINIFile(); // aus Applaunc.ini im Startverzeichnis.
  UpdateAppButtons();
  DragAcceptFiles(true);
}
```

Siehe auch *TFrameWindow::SetupWindow, TComboBox::SetupWindow, TWindow::CleanupWindow*

Antworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_WM_CREATE	EvCreate
EV_WM_CLOSE	EvClose
EV_WM_DESTROY	EvDestroy
EV_WM_SIZE	EvSize
EV_WM_MOVE	EvMove
EV_WM_NCDESTROY	EvNcDestroy
EV_WM_QUERYENDSESSION	EvQueryEndSession
EV_WM_COMPAREITEM	EvCompareItem
EV_WM_DELETEITEM	EvDeleteItem
EV_WM_DRAWITEM	EvDrawItem
EV_WM_MEASUREITEM	EvMeasureItem
EV_WM_CHILDINVALID	EvChildInvalid
EV_WM_VSCROLL	EvVScroll
EV_WM_HSCROLL	EvHScroll
EV_WM_PAINT	EvPaint
EV_WM_SETCURSOR	EvSetCursor
EV_WM_LBUTTONDOWN	EvLButtonDown
EV_COMMAND(CM_EXIT, CmExit)	CmExit
EV_WM_SYSCOLORCHANGE	EvSysColorChange
EV_WM_KILLFOCUS	EvKillFocus
EV_WM_ERASEBKGND	EvEraseBkgnd
EV_MESSAGE(WM_CTLCOLORMSGBOX, EvWin32CtlColor)	EvWin32CtlColor*
EV_MESSAGE(WM_CTLCOLOREDIT, EvWin32CtlColor)	EvWin32CtlColor*
EV_MESSAGE(WM_CTLCOLORLISTBOX, EvWin32CtlColor)	EvWin32CtlColor*
EV_MESSAGE(WM_CTLCOLORBTN, EvWin32CtlColor)	EvWin32CtlColor*
EV_MESSAGE(WM_CTLCOLORDLG, EvWin32CtlColor)	EvWin32CtlColor*
EV_MESSAGE(WM_CTLCOLORSCROLLBAR, EvWin32CtlColor)	EvWin32CtlColor*
EV_MESSAGE(WM_CTLCOLORSTATIC, EvWin32CtlColor)	EvWin32CtlColor*

* Diese Antworttabelleneinträge sind nur unter der Win32-API verfügbar.

Klasse TWindowDC dc.h

TWindowDC ist eine von *TDC* abgeleitete DC-Klasse, die Zugriff auf den gesamten Bereich ermöglicht, den ein Fenster besitzt.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TWindowDC(HWND wnd);
```

Erzeugt ein *TWindow*-Objekt mit dem gegebenen Fenster, dessen Besitzer es ist. Das Datenelement *Wnd* wird auf *wnd* gesetzt.

Siehe auch *TWindowDC::Wnd*

Destruktor

```
~TWindowDC();
```

Löscht dieses Objekt.

Geschützte Konstruktoren

Konstruktor

```
TWindowDC();
```

Wird nur für abgeleitete Klassen verwendet.

Geschütztes Datenelement

Wnd

Enthält ein Handle für das Fenster, dessen Besitzer dieses DC ist.

Siehe auch *TWindowDC::TWindowDC*

Klasse TWindowView docview.h

TWindowView, abgeleitet aus *TWindow* und *TVie*w, ist eine streamfähige Basisklasse, die zur Ableitung auf Fenstern basierender Views verwendet werden kann. Die Funktionen von *TWindowView* setzen die virtuelle Funktion von *TView* außer Kraft, um eine eigene Implementierung zu liefern. Durch das Ableiten einer Klasse von Fenster-Views aus *TWindow* und *TView* fügen Sie dem View Ihres Dokuments die Funktionalität von Fenstern hinzu.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TWindowView (TDocument& doc, TWindow* parent = 0);
```

Erstellt ein mit dem View des Fensters verknüpftes *TWindowView*-Oberflächenobjekt. Setzt *ViewId* auf *NextViewId*. Ruft das *AttachView* des verknüpften Dokuments auf, um das View mit dem Dokument zu verbinden.

Destruktor

```
~TWindowView()
```

Löscht ein *TWindowView*-Objekt und ruft die *DetachView*-Funktion des verbundenen Dokuments auf, um das View von diesem zu trennen.

Öffentliche Elementfunktionen

CanClose

```
bool CanClose();
```

Setzt *TView::CanClose* außer Kraft und liefert Nicht-Null zurück, wenn das Fenster geschlossen werden kann. Prüft sämtliche *CanClose*-Funktionen des verknüpften Dokuments, die alle einen Wert ungleich 0 zurückliefern müssen, ehe die Fenster-View geschlossen werden kann.

Siehe auch *TWindow::CanClose*

GetViewName

```
const char far* GetViewName();
```

Überschreibt *TView::GetViewName* und liefert *StaticName*, den Namen der View, zurück.

Siehe auch *TView::GetViewName*

GetWindow

```
TWindow* GetWindow()
```

Überschreibt *TView::GetWindow* und liefert das *TWindowView*-Objekt als *TWindow* zurück.

Siehe auch *TEditView::GetWindow*, *TView::GetWindow*

SetDocTitle

```
bool SetDocTitle(const char far* docname, int index)
```

Überschreibt *TView::SetDocTitle* und speichert den Titel des Dokuments. Dieser Name wird auf der übergeordneten Chain weitergeleitet, bis ein *TFrameWindow*-Objekt die Daten akzeptiert und in seiner Titelleiste anzeigt.

Siehe auch *TView::SetDocTitle*, *TWindow::SetDocTitle*

StaticName

```
static const char far* StaticName();
```

Liefert »Window View«, den beschreibenden Namen der View, zurück. Dieser Titel wird im Rahmenfeld der Benutzerschnittstelle angezeigt.

Antworttabelleneinträge

Antworttabelleneintrag	Elementfunktion
EV_VN_ISWINDOW	VnIsWindow

Klasse TWindow::TXWindow window.h

TXWIndow ist eine geschachtelte Klasse, die eine aus dem Versuch der Erstellung eines ungültigen Fensters entstandene Exception beschreibt.

Öffentliche Konstruktoren

Konstruktor

```
TXWindow(TWindow* win = 0, uint resourceId = IDS_INVALIDWINDOW);
```

Erstellt ein *TXWindow*-Objekt mit einer Standard-Ressourcen-ID von IDS_INVALIDWINDOW.

Konstruktor

```
TXWindow(const TXWindow& src);
```

Konstruiert ein *TXWindow*-Objekt mit dem Fenster, das fehlgeschlagen ist.

Öffentliche Datenelemente

Window

```
TWindow* Window;
```

Zeigt auf das mit der Exception verknüpfte Fensterobjekt.

Öffentliche Elementfunktionen

Clone

```
TXOwl* Clone();
```

Legt eine Kopie des Exception-Objektes an. *Clone* muß in jeder von *TXOwl* abgeleiteten Klasse implementiert sein.

Msg

```
static string Msg(TWindow*, uint resourceid);
```

Wandelt die Ressourcen-ID in einen String um und liefert die Stringmeldung zurück.

Throw

```
inline void Throw();
```

Löst ein Exception-Objekt aus. *Throw* muß in jeder von *TXOwl* abgeleiteten Klasse implementiert sein.

Unhandled

```
int Unhandled(TModule* app, unsigned promptResId);
```

Wenn eine in der Botschaftsschleife aufgefangene Exception nicht behandelt worden ist, wird *Unhandled* aufgerufen. *Unhandled* löscht das Fenster. Dieser Exception-Typ könnte vorkommen, wenn ein Fenster nicht erzeugt werden kann.

Klasse TXCompatibility except.h

Beschreibt eine Exception, die daher rührt, daß *TModule::Status* auf Nicht-Null gesetzt wurde. Diese Exception wurde aufgenommen, um rückgerichtete Kompatibilität mit ObjectWindows 1.0 zu gewährleisten.

Öffentliche Konstruktoren

Konstruktor

```
TXCompatibility(int statusCode);
```

Erstellt ein *TXCompatibility*-Objekt.

Öffentliche Elementfunktionen

MapStatusCodeToString

```
static string MapStatusCodeToString(int statusCode);
```

Ermittelt den Status-Code von *Tmodule* und überträgt ihn auf einen String.

Klasse TXMenu except.h

Eine eingebettete Klasse mit Namen *TXMenu* beschreibt eine Exception, die auftritt, wenn ein Menüeintrag nicht konstruiert werden kann.

Öffentliche Konstruktoren

Konstruktor

```
TXMenu(unsigned resId = IDS_GDIFAILURE);
```

Konstruiert ein TXMenu-Exception-Objekt mit einer Standardbotschaft IDS_GDIFAILURE.

Öffentliche Elementfunktionen

Throw

```
TXOwl* Clone();
```

Legt eine Kopie des Exception-Objektes an.

Clone

```
void Throw();
```

Löst eine *TXMenu*-Exception aus.

Klasse TXOutOfMemory except.h

Diese Klasse beschreibt eine Exception, die aus einem Speicherüberlauf resultiert.

Öffentliche Konstruktoren

TXOutOfMemory

```
TXOutOfMemory();
```

Konstruiert ein *TXOutOfMemory*-Objekt.

Öffentliche Elementfunktionen

Clone

```
TXOwl* Clone();
```

Legt eine Kopie des Exception-Objektes an. *Clone* muß in jeder von *TXOwl* abgeleiteten Klasse implementiert sein.

Throw

```
void Throw();
```

Löst ein Exception-Objekt aus. *Throw* muß in jeder von *TXOwl* abgeleiteten Klasse implementiert sein.

Klasse TXOwl except.h

TXOwl ist eine übergeordnete Klasse, die zur Beschreibung von Exceptions dient. Es handelt sich dabei um anormale Bedingungen, die außerhalb des Einflußbereichs des Programms liegen. Anstatt diese Klasse direkt zu verwenden, werden Sie in den meisten Fällen eine neue Klasse von *TXOwl* ableiten. Die von *TXOwl* abgeleiteten ObjectWindows-Klassen sind:

TXCompatibility **-** Beschreibt eine Exception, die auftritt, wenn *TModule::Status* nicht Null ist.

TValidator::TXValidator **-** Beschreibt eine Exception, die auftritt, wenn ein ungültiger Validierungs-Ausdruck vorhanden ist.

TWindow::XWindow **-** Beschreibt eine Exception, die das Ergebnis des Versuchs ist, ein ungültiges Fenster zu erzeugen.

TGdiObject::TXGdi **-** Beschreibt eine Exception, die das Ergebnis des Versuchs ist, ein ungültiges GDI zu erzeugen.

TApplication::TXInvalidMainModule **-** Beschreibt eine Exception, die das Ergebnis des Versuchs ist, ein ungültiges Hauptfenster zu erzeugen.

TMenu::XMenu **-** Beschreibt eine Exception, die auftritt, wenn ein Menüobjekt ungültig ist.

TModule::XInvalidModule**-** Beschreibt eine Exception, die auftritt, wenn ein *TModule*-Objekt ungültig ist.

TPrinter::TXPrinter **-** Beschreibt eine Exception, die auftritt, wenn ein Drucker-Gerätekontext ungültig ist.

TXOurOfMemory **-** Beschreibt eine Exception, die bei einem Fehler aufgrund ungenügenden Arbeitsspeichers auftritt.

Jede der Exception-Klassen beschreibt eine spezielle Art von Exception. Wenn Ihr Programm einer gegebenen Situation begegnet, die möglicherweise diese Exception auslösen wird, überträgt es die Steuerung auf das spezifizierte Exception-Behandlungsobjekt. Wenn Sie in ihrem Programmcode Exceptions berücksichtigen, können Sie es vermeiden, überall Prozeduren für die Fehlerbehandlung darin zu verteilen.

Setzen Sie, um eine Exception-Behandlung zu erstellen, das Schlüsselwort try vor den Programmblock, der die anormale Bedingung herbeiführen könnte (der Programmteil, der ein Exception-Objekt erzeugen könnte) und das Schlüsselwort catch vor den dem try-Block folgenden Block. Wird eine Exception innerhalb des try-Blocks ausgelöst, werden die Klassen innerhalb jeder der nachfolgenden catch-Klauseln der Reihe nach durchsucht. Die erste, die der Klasse des Exception-Objektes entspricht, wird ausgeführt.

Der folgende Ausschnitt aus MDIFILE.CPP, einem Programm auf Ihrer Beispieldiskette, zeigt, wie ein try/catch-Block um den Programmtext gesetzt wird, der eine Exception auslösen könnte.

```
void TMDIFileApp::CmRestoreState()
{
  char* errorMsg = 0;

  ifpstream is(DskFile);
  if (is.bad())
    errorMsg = "Kann die Datei nicht öffnen.";

// try-Block //

  else {
    if (Client CloseChildren()) {
      try {
        is >* this;
        if (is.bad())
          errorMsg = "Fehler beim Lesen der Datei.";
        else
          ClientCreateChildren();
      }

// catch-Block //

      catch (xalloc) {
        ClientCloseChildren();
        errorMsg = "Nicht genügend Speicher, um Datei zu öffnen.";
      }
    }
  }
  if (errorMsg)
```

```
    MainWindowMessageBox(errorMsg, "Fehler", MB_OK | MB_ICONEXCLAMATION);
}
```

Siehe auch *TXBase*

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TXOwl(const string& msg, unsigned ResId = 0);
```

Lädt die String-Ressource, die durch den Parameter *resId* festgelegt wird und verwendet diese Ressource, um das *TXBase*-Objekt zu initialisieren.

```
TXOwl(unsigned resId, TModule* module = ::module);
```

Erstellt ein *TXOwl*-Objekt mit einer Ressourcen-ID.

Destruktor

```
~TXOwl();
```

Entfernt ein *TXOwl*-Objekt.

Öffentliche Elementfunktionen

Clone

```
TXOwl* Clone();
```

Erstellt eine Kopie des Exception-Objektes. *Clone* muß in jeder von *TXOwl* abgeleiteten Klasse implementiert sein.

GetErrorCode

```
unsigned GetErrorCode () const;
```

Gibt die Ressourcen-ID zurück.

ResourceIdToString

```
static string ResourceIdToString(bool* found, unsigned resId, TModule*
module = ::module);
```

Wandelt eine Ressourcen-ID in einen String um und liefert einen den Exception-Zustand bezeichnenden String zurück. Wenn die String-Meldung nicht geladen werden kann, wird die Meldung »not found« zurückgeliefert. Setzt den Parameter *found* auf **true**, wenn die Ressource gefunden wurde. Andernfalls wird *found* auf **false** gesetzt.

Throw

```
void Throw();
```

Löst ein Exception-Objekt aus. *Throw* muß in jeder von *TXOwl* abgeleiteten Klasse implementiert sein.

Unhandled

```
virtual int Unhandled(TModule* app, unsigned promptResId);
```

Unhandled wird aufgerufen, wenn in der Hauptebene der Botschaftsschleife eine unbearbeitete Exception abgefangen wird.

vnxxxx View-Benachrichtigungskonstanten docview.h

Die *View-Benachrichtigungskonstanten* werden verwendet, um das View von einem gegebenen Ereignis zu benachrichtigen.

Tabelle 2.36: Vnxxxx View-Benachrichtigungs-IDs

Konstante	Bedeutung
vnCommit	Es werden Änderungen an das Dokument übergeben.
vnCustomBase	Basisereignis für Dokumentbenachrichtigungen.
vnDocOpened	Das Dokument wurde geöffnet.
vnDocClosed	Das Dokument wurde geschlossen.
vnIsDirty	Ist **true**, wenn nicht-übergebene Botschaften vorliegen.
vnIsWindow	Ist **true**, wenn das übergebene HWND zu diesem View gehört.
vnRevert	Die vorherigen Daten des Dokuments werden neu geladen und die aktuellen Daten des View damit überschrieben.
vnViewOpened	Ein neues View wurde erstellt.
vnViewClosed	Ein View wird in Kürze gelöscht.

Siehe auch *TListView*, *TEditView*

K A P I T E L 3

Die Ereignisbehandlungsroutinen von ObjectWindows

Dieses Kapitel umfaßt mehrere Tabellen, die vordefinierte ObjectWindows-Antworttabellenmakros und Ereignisbehandlungsfunktionen auflisten. Jede Tabelle verzeichnet den Namen des ObjectWindows-Makros, alle erforderlichen Makroargumente und die assoziierte Ereignisbehandlungsfunktion. Die Tabellen sind auf folgende Weise organisiert:

Diese Schaltermakros behandeln BN_xxxx-Benachrichtigungs-Codes. Um den Namen des Benachrichtigungs-Codes zu erhalten, der dem EV_XXXX-Makro entspricht, entfernen Sie das EV_Präfix.

Tabelle 3.1: Schalter-Benachrichtigungsbotschaften

Makro	Makroargumente	Deklaration der Antwortfunktion
EV_BN_CLICKED	ID, UserName	void UserName()
EV_BN_DISABLE	ID, UserName	void UserName()
EV_BN_DOUBLECLICKED	ID, UserName	void UserName()
EV_BN_HILITE	ID, UserName	void UserName()
EV_BN_PAINT	ID, UserName	void UserName()
EV_BN_UNHILITE	ID, UserName	void UserName()

Die folgenden Makros behandeln Botschaften, die ein untergeordnetes Fenster an sein übergeordnetes Fenster schickt:

Tabelle 3.2: Benachrichtigungsbotschaften, die auf der ID untergeordneter Fenster basieren

Makro	Makroargumente	Deklaration der Antwortfunktion
EV_CHILD_NOTIFY_AND_CODE	ID, Code, *UserName*	void *UserName*(WPARAM)
EV_CHILD_NOTIFY_ALL_CODES	ID, *UserName*	void *UserName*(uint)
EV_CHILD_NOTIFY_AT_CHILD	Code, *UserName*	void *UserName*()
EV_CHILD_NOTIFY	ID, Code, *UserName*	void *UserName*()

Die folgenden Kombinationsfenstermakros behandeln CBN_xxxx-Benachrichtigungs-Codes. Um den Namen des Benachrichtigungs-Codes, der dem EV_XXXX-Makro entspricht, zu bestimmen, müssen Sie nur das Präfix EV_ entfernen.

Tabelle 3.3: Kombinationsfenster-Benachrichtigungsbotschaften

Makro	Makroargumente	Deklaration der Antwortfunktion
EV_CBN_CLOSEUP	ID, *UserName*	void *UserName*()
EV_CBN_DBLCLK	ID, *UserName*	void *UserName*()
EV_CBN_DROPDOWN	ID, *UserName*	void *UserName*()
EV_CBN_EDITCHANGE	ID, *UserName*	void *UserName*()
EV_CBN_EDITUPDATE	ID, *UserName*	void *UserName*()
EV_CBN_ERRSPACE	ID, *UserName*	void *UserName*()
EV_CBN_KILLFOCUS	ID, *UserName*	void *UserName*()
EV_CBN_SELCHANGE	ID, *UserName*	void *UserName*()
EV_CBN_SELENDCANCEL	ID, *UserName*	void *UserName*()
EV_CBN_SELENDOK	ID, *UserName*	void *UserName*()
EV_CBN_SETFOCUS	ID, *UserName*	void *UserName*()

Die folgenden Makros behandeln WM_COMMAND-Botschaften:

Tabelle 3.4: Kommandobotschaften

Makro	Makroargumente	Deklaration der Antwortfunktion
EV_COMMAND	CMD ID, *UserName*	void *UserName*()
EV_COMMAND_AND_ID	CMD ID, *UserName*	void *UserName*(WPARAM)
EV_COMMAND_ENABLE	CMD ID, *UserName*	void *UserName*(TCommandEnabler&)

Die folgenden Makros behandeln Botschaften, die von der Dokumentverwaltung generiert wurden.

Tabelle 3.5: Dokumentverwaltungsbotschaften

Makro	Makroargumente	Deklaration der Antwortfunktion
EV_OWLDOCUMENT	ID, *UserName*	void *UserName*(TDocument& document)
EV_OWLNOTIFY	ID, *UserName*	bool *UserName*(LPARAM&)
EV_OWLVIEW	ID, *UserName*	void *UserName*(TView& view)

Die folgenden Makros behandeln View-bezogene Botschaften, die vom Dokumentverwalter generiert wurden: *VnHandler* ist ein generischer Begriff für die Behandlungsfunktion für View-Benachrichtigungen.

Tabelle 3.6: Dokument-View-Botschaften

Makro	Deklaration der Antwortfunktion
EV_VN_VIEWOPENED	bool VnViewOpened(TView* view)
EV_VN_VIEWCLOSED	bool VnViewClosed(TView* view)

EV_VN_DOCOPENED	bool VnDocOpened(int openMode)
EV_VN_DOCCLOSED	bool VnDocClosed(int openMode)
EV_VN_COMMIT	bool VnCommit(bool force)
EV_VN_REVERT	bool VnRevert(bool clear)
EV_VN_ISDIRTY	bool VnIsDirty()
EV_VN_ISWINDOW	bool VnIsWindow(HWND hWnd)

Die folgenden Editierfeldmakros behandeln EN_xxxx-Benachrichtigungs-Codes. Um den Namen des Benachrichtigungs-Code, der dem EV_XXXX-Makro entspricht, zu bestimmen, müssen Sie nur das Präfix EV_ entfernen.

Tabelle 3.7: Editierfeld-Benachrichtigungsbotschaften

Makro	Makroargumente	Deklaration der Antwortfunktion
EV_EN_CHANGE	ID, *UserName*	void *UserName*()
EV_EN_ERRSPACE	ID, *UserName*	void *UserName*()
EV_EN_HSCROLL	ID, *UserName*	void *UserName*()
EV_EN_KILLFOCUS	ID, *UserName*	void *UserName*()
EV_EN_MAXTEXT	ID, *UserName*	void *UserName*()
EV_EN_SETFOCUS	ID, *UserName*	void *UserName*()
EV_EN_UPDATE	ID, *UserName*	void *UserName*()
EV_EN_VSCROLL	ID, *UserName*	void *UserName*()

Die folgenden Listenfeldmakros behandeln LBN_xxxx-Benachrichtigungs-Codes. Um den Namen des Benachrichtigungs-Code, der dem EV_XXXX-Makro entspricht, zu bestimmen, müssen Sie nur das Präfix EV_ entfernen.

Tabelle 3.8: Listenfeld-Benachrichtigungsbotschaften

Makro	Makroargumente	Deklaration der Antwortfunktion
EV_LBN_DBLCLK	ID, *UserName*	void *UserName*()
EV_LBN_ERRSPACE	ID, *UserName*	void *UserName*()
EV_LBN_KILLFOCUS	ID, *UserName*	void *UserName*()
EV_LBN_SELCANCEL	ID, *UserName*	void *UserName*()
EV_LBN_SELCHANGE	ID, *UserName*	void *UserName*()
EV_LBN_SETFOCUS	ID, *UserName*	void *UserName*()

Diese Makros behandeln ObjectComponents-Botschaften.

Tabelle 3.9: ObjectComponents-Botschaften

Makro	Deklaration der Antwortfunktion
EV_OC_VIEWPARTINVALID	bool EvOcViewPartInvalid(TOcPart far&)
EV_OC_VIEWTITLE	const char far* EvOcViewTitle()
EV_OC_VIEWBORDERSPACEREQ	EvOcViewBorderSpaceReq
EV_OC_VIEWBORDERSPACESET	bool EvOcViewBorderSpaceSet(TRect far&)

EV_OC_VIEWDROP	bool EvOcViewDrop (TOcDragDrop far&)
EV_OC_VIEWDRAG	bool EvOcViewDrag(TOcDragDrop far&)
EV_OC_VIEWSCROLL	bool EvOcViewScroll (TOcScrollDir)
EV_OC_VIEWGETSCALE	bool EvOcViewGetScale(TOcScaleFactor&)
EV_OC_VIEWGETSITERECT	bool EvOcViewGetSiteRect(TRect far*)
EV_OC_VIEWSETSITERECT	bool EvOcViewSetSiteRect(TRect far*)
EV_OC_VIEWPAINT	bool EvOcViewPaint (TOcViewPaint far&)
EV_OC_VIEWSAVEPART	bool EvOcViewSavePart(TOcSaveLoad far& ocSave)
EV_OC_VIEWLOADPART	bool EvOcViewLoadPart(TOcSaveLoad far& ocLoad)
EV_OC_VIEWINSMENUS	bool EvOcViewInsMenus (TOcMenuDescr far&)
EV_OC_VIEWSHOWTOOLS,	bool EvOcViewShowTools(TOcToolBarInfo far& tbi)
EV_OC_VIEWGETPALETTE	bool EvOcViewGetPalette (LOGPALETTE far*)
EV_OC_VIEWCLIPDATA,	HANDLE EvOcViewClipData(TOcFormat far&)
EV_OC_VIEWCLOSE	bool EvOcViewClose ()
EV_OC_VIEWPARTSIZE	bool EvOcViewPartSize(TRect far&)
EV_OC_VIEWOPENDOC	bool EvOcViewOpenDoc(const char far&)
EV_OC_VIEWATTACHWINDOW	bool EvOcViewAttachWindow()
EV_OC_VIEWSETSCALE	bool EvOcViewSetScale(TOcScaleFactor&)
EV_OC_APPINSMENUS	bool EvOcAppInsMenus(TOcMenuDescr far&)
EV_OC_APPMENUS	bool EvOcAppMenus(TOcMenuDescr far&)
EV_OC_APPPROCESSMSG	bool EvOcAppProcessMsg(MSG far*)
EV_OC_APPFRAMERECT	bool EvOcAppFrameRect(TRect far*)
EV_OC_APPBORDERSPACEREQ	bool EvOcAppBorderSpaceReq(TRect far*)
EV_OC_APPBORDERSPACESET	bool EvOcAppBorderSpaceSet(TRect far*)
EV_OC_APPSTATUSTEXT	void EvOcAppStatusText(const char far*)
EV_OC_APPRESTOREUI	void EvOcAppRestoreUI()
EV_OC_APPDIALOGHELP	void EvOcAppDialogHelp(TOcDialogHelp far& dh)
EV_OC_APPSHUTDOWN	void EvOcAppShutDown()

Die folgenden Bildlaufleisten-Makros behandeln SB_xxxx-Benachrichtigungs-Codes. Um den Namen des Benachrichtigungs-Codes zu erhalten, der dem EV_XXXX-Makro entspricht, entfernen Sie das EV_Präfix.

Tabelle 3.10: Bildlaufleisten-Benachrichtigungsbotschaften

Makro	Makroargumente	Deklaration der Antwortfunktion
EV_SB_LINEDOWN	ID, UserName	void UserName()
EV_SB_LINEUP	ID, UserName	void UserName()
EV_SB_PAGEDOWN	ID UserName	void UserName()
EV_SB_PAGEUP	ID, UserName	void UserName()
EV_SB_TOP	ID, UserName	void UserName()
EV_SB_BOTTOM	ID, UserName	void UserName()
EV_SB_THUMBPOSITION	ID, UserName	void UserName()
EV_SB_ENDSCROLL	ID, UserName	void UserName()
EV_SB_BEGINTRACK	ID, UserName	void UserName()

Tabelle 3.11 listet die Makros auf, die Windows-Botschaften behandeln. Diese Makros sind in windowev.h definiert. Um den Namen der Windows-Botschaft zu bestimmen, die dem EV_XXXX-Makro entspricht,

müssen Sie nur das Präfix EV_ entfernen. WM_ACTIVATE ist zum Beispiel der Name der Windows-Botschaft, die das Makro EV_WM_ACTIVATE behandelt. Diejenigen Makros, welche die Standardbotschaften von Windows extrahieren (d.h. Aufbrechen der Parameter LPARAM und WPARAM in separate Bestandteile), erwarten keine Argumente. Sie übergeben die extrahierten Parameter direkt an die vordefinierte EVxxxx-Botschaftsfunktion. Die Windows-Standardbotschaften sind in Ihrer Windows-Dokumentation beschrieben.

Tabelle 3.11:Windows-Standardbotschaften

Makro	Deklaration der Antwortfunktion
EV_WM_ACTIVATE	void EvActivate(uint active, bool minimized, HWND hWndOther)
EV_WM_ACTIVATEAPP	void EvActivateApp(bool active, HANDLE threadId)[2]
	void EvActivateApp(bool active, HTASK hTask)[1]
EV_WM_ASKCBFORMATNAME	void EvAskCBFormatName(uint bufLen, char far* buffer)
EV_WM_CANCELMODE	void EvCancelMode()
EV_WM_CHANGECBCHAIN	void EvChangeCBChain(uint bufLen, char far* buffer)
EV_WM_CHAR	void EvChar(uint key, uint repeatCount, uint flags)
EV_WM_CHARTOITEM	int EvCharToItem(uint key, HWND hWndListBox, uint caretPos)
EV_WM_CHILDACTIVATE	void EvChildActivate()
EV_WM_CHILDINVALID	void EvChildInvalid(HWND)
EV_WM_CLOSE	void EvClose()
EV_WM_COMPACTING	void EvCompacting(uint compactRatio)
EV_WM_COMPAREITEM	LRESULT EvCompareItem(uint ctrlId, COMPAREITEMSTRUCT far& compareInfo)
EV_WM_CREATE	int EvCreate(CREATESTRUCT far &)
EV_WM_CTLCOLOR	HBRUSH EvCtlColor(HDC, HWND hWndChild, uint ctlType)
EV_WM_DEADCHAR	void EvDeadChar(uint deadKey, uint repeatCount, uint flags)
EV_WM_DELETEITEM	void EvDeleteItem(uint ctrlId, DELETEITEMSTRUCT far& deleteInfo)
EV_WM_DESTROY	void EvDestroy()
EV_WM_DESTROYCLIPBOARD	void EvDestroyClipboard()
EV_WM_DEVMODECHANGE	void EvDevModeChange(char far* devMode)
EV_WM_DRAWCLIPBOARD	void EvDrawClipboard()
EV_WM_DRAWITEM	void EvDrawItem(uint ctrlId, DRAWITEMSTRUCT far& drawInfo)
EV_WM_DROPFILES	void EvDropFiles(TDropInfo dropInfo)
EV_WM_ENABLE	void EvEnable(bool enabled)
EV_WM_ENDSESSION	void EvEndSession(bool endSession)
EV_WM_ENTERIDLE	void EvEnterIdle(uint source, HWND hWndDlg)
EV_WM_ERASEBKGND	bool EvEraseBkgnd(HDC)
EV_WM_FONTCHANGE	void EvFontChange()
EV_WM_GETDLGCODE	uint EvGetDlgCode()
EV_WM_GETFONT	HFONT EvGetFont();
EV_WM_GETMINMAXINFO	void EvGetMinMaxInfo(MINMAXINFO far &)
EV_WM_GETTEXT	void EvGetText(uint bufLen, char far* buffer)
EV_WM_GETTEXTLENGTH	uint EvGetTextLength()
EV_WM_HOTKEY	void EvHotKey(int idHotKey)[2]
EV_WM_HSCROLL	void EvHScroll(uint scrollCode, uint thumbPos, HWND hWndCtl)
EV_WM_HSCROLLCLIPBOARD	void EvHScrollClipboard(HWND hWndCBViewer, uint scrollCode, uint pos)

EV_WM_ICONERASEBKGND	void EvIconEraseBkgnd(HDC)
EV_WM_INITMENU	void EvInitMenu(HMENU)
EV_WM_INITMENUPOPUP	void EvInitMenuPopup(HMENU hPopupMenu, uint index, bool sysMenu)
EV_WM_INPUTFOCUS	void EvInputFocus(bool gainingFocus)[2]
EV_WM_KEYDOWN	void EvKeyDown(uint key, uint repeatCount, uint flags)
EV_WM_KEYUP	void EvKeyUp(uint key, uint repeatCount, uint flags)
EV_WM_KILLFOCUS	void EvKillFocus(HWND hWndGetFocus)
EV_WM_LBUTTONDBLCLK	void EvLButtonDblClk(uint modKeys, TPoint& point)
EV_WM_LBUTTONDOWN	void EvLButtonDown(uint modKeys, TPoint& point)
EV_WM_LBUTTONUP	void EvLButtonUp(uint modKeys, TPoint& point)
EV_WM_MBUTTONDBLCLK	void EvMButtonDblClk(uint modKeys, TPoint& point)
EV_WM_MBUTTONDOWN	void EvMButtonDown(uint modKeys, TPoint& point)
EV_WM_MBUTTONUP	void EvMButtonUp(uint modKeys, TPoint& point)
EV_WM_MDIACTIVATE	void EvMDIActivate(HWND hWndActivated, HWND hWndDeactivated)
EV_WM_MDICREATE	LRESULT EvMDICreate(MDICREATESTRUCT far& createStruct)
EV_WM_MDIDESTROY	void EvMDIDestroy(HWND hWnd);
EV_WM_MENUCHAR	uint EvMenuChar(uint nChar, uint menuType, HMENU hMenu)
EV_WM_MENUSELECT	void EvMenuSelect(uint menuItemId, uint flags, HMENU hMenu)
EV_WM_MEASUREITEM	void EvMeasureItem(uint ctrlId, MEASUREITEMSTRUCT far& measureInfo)
EV_WM_MOUSEACTIVATE	uint EvMouseActivate(HWND hWndTopLevel, uint hitTestCode, uint msg)
EV_WM_MOUSEMOVE	void EvMouseMove(uint modKeys, TPoint& point)
EV_WM_MOVE	void EvMove(TPoint &clientOrigin)
EV_WM_NCACTIVATE	bool EvNCActivate(bool active)
EV_WM_NCCALCSIZE	uint EvNCCalcSize(bool calcValidRects, NCCALCSIZE_PARAMS far &)
EV_WM_NCCREATE	bool EvNCCreate(CREATESTRUCT far &)
EV_WM_NCDESTROY	void EvNCDestroy()
EV_WM_NCHITTEST	uint EvNCHitTest(TPoint& point)
EV_WM_NCLBUTTONDBLCLK	void EvNCLButtonDblClk(uint hitTest, TPoint& point)
EV_WM_NCLBUTTONDOWN	void EvNCLButtonDown(uint hitTest, TPoint& point)
EV_WM_NCLBUTTONUP	void EvNCLButtonUp(uint hitTest, TPoint& point)
EV_WM_NCMBUTTONDBLCLK	void EvNCMButtonDblClk(uint hitTest, TPoint& point)
EV_WM_NCMBUTTONDOWN	void EvNCMButtonDown(uint hitTest, TPoint& point)
EV_WM_NCMBUTTONUP	void EvNCMButtonUp(uint hitTest, TPoint& point)
EV_WM_NCMOUSEMOVE	void EvNCMouseMove(uint hitTest, TPoint& point)
EV_WM_NCPAINT	void EvNCPaint()
EV_WM_NCRBUTTONDBLCLK	void EvNCRButtonDblClk(uint hitTest, TPoint& point)
EV_WM_NCRBUTTONDOWN	void void EvNCRButtonDown(uint hitTest, TPoint& point)
EV_WM_NCRBUTTONUP	void EvNCRButtonUp(uint hitTest, TPoint& point)
EV_WM_OTHERWINDOWCREATED	void EvOtherWindowCreated(HWND hWndOther)[2]
EV_WM_OTHERWINDOWDESTROYED	void EvOtherWindowDestroyed(HWND hWndOther)[2]
EV_WM_PAINT	void EvPaint()
EV_WM_PAINTCLIPBOARD	void EvPaintClipboard(HWND, HANDLE hPaintStruct)
EV_WM_PAINTICON	void EvPaintIcon()[2]
EV_WM_PALETTECHANGED	void EvPaletteChanged(HWND hWndPalChg)
EV_WM_PALETTEISCHANGING	void EvPaletteIsChanging(HWND hWndPalChg)
EV_WM_PARENTNOTIFY	void EvParentNotify(uint event, uint childHandleOrX, uint childIDOrY)
EV_WM_POWER	int EvPower(uint powerEvent)
EV_WM_QUERYDRAGICON	HANDLE EvQueryDragIcon()
EV_WM_QUERYENDSESSION	bool EvQueryEndSession()
EV_WM_QUERYNEWPALETTE	bool EvQueryNewPalette()

EV_WM_QUERYOPEN	bool EvQueryOpen()
EV_WM_RBUTTONDBLCLK	void EvRButtonDblClk(uint modKeys, TPoint& point)
EV_WM_RBUTTONDOWN	void EvRButtonDown(uint modKeys, TPoint& point)
EV_WM_RBUTTONUP	void EvRButtonUp(uint modKeys, TPoint& point)
EV_WM_RENDERALLFORMATS	void EvRenderAllFormats()
EV_WM_RENDERFORMAT	void EvRenderFormat(uint dataFormat)
EV_WM_SETCURSOR	bool EvSetCursor(HWND hWndCursor, uint hitTest, uint mouseMsg)
EV_WM_SETFOCUS	void EvSetFocus(HWND hWndLostFocus)
EV_WM_SETFONT	void EvSetFont(HFONT)
EV_WM_SETTEXT	void EvSetText(char far* text)
EV_WM_SHOWWINDOW	void EvShowWindow(bool show, uint status)
EV_WM_SIZE	void EvSize(uint sizeType, TSize& size)
EV_WM_SIZECLIPBOARD	void EvSizeClipboard(HWND hWndViewer, HANDLE hRect)
EV_WM_SPOOLERSTATUS	void EvSpoolerStatus(uint jobStatus, uint jobsLeft)
EV_WM_SYSCHAR	void EvSysChar(uint key, uint repeatCount, uint flags)
EV_WM_SYSCOLORCHANGE	void EvSysColorChange()
EV_WM_SYSCOMMAND	void EvSysCommand(uint cmdType, TPoint& point)
EV_WM_SYSDEADCHAR	void EvSysDeadChar(uint key, uint repeatCount, uint flags)
EV_WM_SYSKEYDOWN	void EvSysKeyDown(uint key, uint repeatCount, uint flags)
EV_WM_SYSKEYUP	void EvSysKeyUp(uint key, uint repeatCount, uint flags)
EV_WM_SYSTEMERROR	void EvSystemError(uint error)
EV_WM_TIMECHANGE	void EvTimeChange()
EV_WM_TIMER	void EvTimer(uint timerId)
EV_WM_VKEYTOITEM	int EvVKeyToItem(uint key, HWND hWndListBox, uint caretPos)
EV_WM_VSCROLL	void EvVScroll(uint scrollCode, uint thumbPos, HWND hWndCtl)
EV_WM_VSCROLLCLIPBOARD	void EvVScrollClipboard(HWND hWndCBViewer, uint scrollCode, uint pos)
EV_WM_WINDOWPOSCHANGED	void EvWindowPosChanged(WINDOWPOS far &windowPos)
EV_WM_WINDOWPOSCHANGING	void EvWindowPosChanging(WINDOWPOS far &windowPos)
EV_WM_WININICHANGE	void EvWinIniChange(char far* section)

[1] nur WIN16-API
[2] nur WIN32-API

Die folgenden Editierfeldmakros behandeln WM_VBXFIREEVENT-Botschaften, die durch VBX-Dialogelemente generiert wurden. *EvHandler* ist ein generischer Begriff für eine spezifische VBX-Dialogelementbotschaft (wie zum Beispiel *EvClick*).

Tabelle 3.12: VBX-Dialogelementbotschaften

Makro	Makroargumente	Deklaration der Antwortfunktion
EV_VBXEVENTNAME	ID, Ereignis, *EvHandler*	void *EvHandler*(VBXEVENT FAR *event)
EV_VBXEVENTINDEX	ID, Ereignis, *EvHandler*	void *EvHandler*(VBXEVENT FAR *event)

Diese Makros behandeln benutzerdefinierte Botschaften:

Tabelle 3.13: Benutzerdefinierte Botschaften

Makro	Makroargumente	Deklaration der Antwortfunktion
EV_MESSAGE	*UserMessage*, *UserName*	LRESULT *UserName*(WPARAM,LPARAM)
EV_REGISTERED	Registrierter Name, *UserName*	LRESULT *UserName*(WPARAM,LPARAM)

Einige Ereignisbehandlungsfunktionen oder -botschaften haben keine vordefinierten Namen. In solchen Fällen wird in diesem Kapitel der generische Begriff *UserName* verwendet. Er zeigt an, daß Sie einen beliebigen Funktionsnamen verwenden können, solange der Funktionskopf mit dem vom Antworttabellenmakro geforderten Kopf übereinstimmt. In ähnlicher Weise zeigt der Begriff *UserMessage* an, daß Sie Ihre eigene Botschafts-ID definieren können.

Für die Beschreibungen dieser Makros und Funktionen gelten hinsichtlich der Auflistung der Argumente der Botschaftsmakros folgende Konventionen:

- *ID* bezieht sich auf die Kennung des untergeordneten Fensters (beispielsweise ID_GROUPBOX)
- *CMD ID* bezieht sich auf die Befehlskennung (beispielsweise CM_FILENEW)
- Code bezieht sich auf den Benachrichtigungscode (beispielsweise BN_CLICKED), der gesendet wird.

K A P I T E L 4

ObjectWindows-Zuteilungsfunktionen

Zuteilungsfunktionen zerlegen die Parameter *lparam* und *wparam* von Windows-Botschaften in ihre jeweiligen Datentypen und übergeben die Kontrolle an eine Elementfunktion von ObjectWindows. Wenn Windows zum Beispiel an eine Anwendung eine WM_CTLCOLOR-Botschaft schickt, ist *wParam* in Wirklichkeit ein HDC und *lParam* hat ein HWND und ein UINT intern versteckt. Nachdem die Zuteilungsfunktion die Parameter *wParam* und *lParam* in ihre Bestandteile zerlegt hat, teilt sie die Botschaft WM_CTLCOLOR von Windows der folgenden Elementfunktion von ObjectWindows zu:

```
HBRUSH EvCtlColor(HDC,HWND,UINT);
```

Obwohl Zuteilungsfunktionen für spezifische Botschaften in der Windows-API geschrieben sind, haben sie keine Kenntnisse über die aktuellen Windows-Botschaften, die sie gerade zuteilen. Sie rufen diese Funktionen niemals direkt auf; stattdessen ruft *TEventHandler::Dispatch* die passende Funktion, basierend auf dem Element *pmf* des Antworttabelleneintrags (Zeiger auf eine Elementfunktion), zur Laufzeit auf.

Die folgenden vier Parameter sind allen Zuteilungsfunktionen gemeinsam:

- *GENERIC& generic* ist der Zeiger auf das Objekt (z.B. *TEdit*).
- *GENERIC::*pmf* ist der Zeiger auf die Elementfunktion (z.B. *EvActivate*).
- WPARAM ist der Botschaftsparameter, den die Zuteilungsfunktion extrahiert.
- LPARAM ist der Botschaftsparameter, den die Zuteilungsfunktion extrahiert.

Der im Antworttabellenmakro verwendete Funktionskopf hängt von der Art des Botschaftsextrahierens, das die Funktion ausführt, ab. Der erste Buchstabe des Kopfes zeigt den Rückgabetyp an (z.B. *U* oder eine vorzeichenlose Ganzzahl). Die zweite Gruppe von Buchstaben legt die Argu-

mente der Funktion fest (z.B. *POINT* für Typ *TPOINT* oder *U* für Typ UINT). Die folgende Tabelle verzeichnet die Abkürzungen für die Funktionsköpfe von Elementfunktionen und ihre entsprechenden Datentypen.

Abkürzung	Datentyp
i	**int**
v	**void**
H	HANDLE (gleiche Größe wie uint)
I32	**int32**
POINT	TPoint& (TPoint-Objekt konstruiert)
POINTER	**void*** (Modell der Systemumgebung)
U	**uint**

Die Namen von Routinen zum Extrahieren von Nachrichten, die der Behandlung benutzerdefinierter Botschaften dienen, basieren auf den jeweils von ihnen zerlegten Botschaften. Die Routine WM_MDIACTIVATE zum Extrahieren von Nachrichten hat beispielsweise folgenden Prototyp:

```
int32_v_MdiActivate_Dispatch(GENERIC& generic, HBRUSH
(GENERIC::*pmf)(uint, uint),
   uint  wParam, int32 lParam);
```

ObjectWindows verwendet dieselben Zuteilungsfunktionen für Botschaften, die dieselbe Art von Extrahieren erfordern. So besitzen zum Beispiel beide Ereignisbehandlungen, WM_HSCROLL und WM_VSCROLL, ein **void** (*)(UINT, UINT,HWND) als Funktionskopf. Die Windows-Botschaft hat ferner die Parameter *wParam* und *lParam* an derselben Stelle. Deshalb verwendet ObjectWindows dieselben Zuteilungsfunktionen für beide Botschaften.

Folgende Aliase wurden zur Sicherstellung der Abwärtskompatibilität definiert:

```
#define HBRUSH_HDC_W_U_Dispatch          U_U_U_U_Dispatch
#define LRESULT_U_U_W_Dispatch           I32_MenuChar_Dispatch
#define LRESULT_WPARAM_LPARAM_Dispatch   I32_WPARAM_LPARAM_Dispatch
#define v_U_B_W_Dispatch                 v_Activate_Dispatch
#define v_W_W_Dispatch                   v_MdiActivate_Dispatch
```

Siehe auch *TEventHandler*

Liste der Zuteilungsfunktionen von ObjectWindows

i_LPARAM_Dispatch

dispatch.h

```
int32 i_LPARAM_Dispatch(GENERIC& generic, int (GENERIC::*pmf)(int32),
uint wParam, int32 lParam);
```

Übergibt *lParam* als **int32** und gibt ein int zurück.

i_U_W_U_Dispatch

dispatch.h

```
int32 i_U_W_U_Dispatch(GENERIC& generic, int (GENERIC::*pmf)(uint, uint,
uint), uint wParam, int32 lParam);
```

Übergibt bei Win32 *wParam.lo* (das niedere Wort von *wParam*) als **uint**, *lParam* als **uint** und *wParam.hi* (das HIWORD von *wParam*) als **uint**, gibt einen **int** zurück. Übergibt bei Win16 *wParam* als **uint**, *lParam.lo* als **uint** und *lParam.hi* als **uint**, gibt einen **int** zurück. Diese teilweise benutzerdefinierte Zuteilungsfunktion wird für WM_CHARTOITEM- und WM_KEYTOITEM-Botschaften verwendet.

i_WPARAM_Dispatch

dispatch.h

```
int32 i_WPARAM_Dispatch(GENERIC& generic, int (GENERIC::*pmf)(uint), uint
wParam, int32 lParam);
```

Übergibt einen *wParam* als **uint** und liefert einen **int** zurück.

I32_Dispatch

dispatch.h

```
int32 I32_Dispatch(GENERIC& generic, int32 (GENERIC::*pmf)(), uint,
int32);
```

Dieses Extraktionsmakro übergibt nichts und liefert einen **int32** zurück.

I32_LPARAM_Dispatch

dispatch.h

```
int32 I32_LPARAM_Dispatch(GENERIC& generic, int32 (GENERIC::*pmf)(int32),
uint, int32 lParam);
```

Dieses Extraktionsmakro übergibt *lParam* als **int32** und liefert einen **int32** zurück.

I32_WPARAM_LPARAM_Dispatch

dispatch.h

```
int32 I32_WPARAM_LPARAM_Dispatch(GENERIC& generic, int32
(GENERIC::*pmf)(uint, int32), uint wParam, int32 lParam);
```

Diese Zuteilungsfunktion übergibt *wParam* als **uint** und *lParam* als **int32**. Es liefert einen **int32** zurück.

I32_MenuChar_Dispatch

dispatch.h

```
int32 I32_MenuChar_Dispatch(GENERIC& generic, uint (GENERIC::*pmf) (uint,
uint, uint), uint wParam, int32 lParam);
```

Extraktionsmakro für WM_PARENTNOTIFY-Botschaften. Übergibt drei **uints** und liefert einen **int32** zurück.Unter Win32 ist der erste **uint** *wParam.lo,* der zweite **uint** *wParam.hi* und der dritte **uint** *lParam*. Unter Win16 ist der erste **uint** *wParam,* der zweite **uint** *lParam.lo* und der dritte **uint** *lParam.hi*.

I32_U_Dispatch

dispatch.h

```
int32 I32_U_Dispatch(GENERIC& generic, int32 (GENERIC::*pmf)(uint), int32
lParam);
```

Dieses Extraktionsmakro übergibt *lParam* und **uint** und liefert einen **int32** zurück.

U_Dispatch

dispatch.h

```
int32 U_Dispatch(GENERIC& generic, uint (GENERIC::*pmf) (), uint wParam,
int32 lParam)
```

Übergibt keine Argumente und liefert einen **uint** zurück.

U_LPARAM_Dispatch

dispatch.h

```
int32 U_LPARAM_Dispatch(GENERIC& generic, uint (GENERIC::*pmf) (int32),
uint wParam, int32 lParam);
```

Übergibt *lParam* als **int32** und liefert einen **uint** zurück.

U_POINT_Dispatch

dispatch.h

```
int32 U_POINT_Dispatch(GENERIC& generic, uint (GENERIC::*pmf) (TPoint&),
uint wParam, int32 lParam);
```

Übergibt *lParam* als *TPoint&* und liefert einen **uint** zurück.

U_POINTER_Dispatch

dispatch.h

```
int32 U_POINTER_Dispatch(GENERIC& generic, uint (GENERIC::*pmf) (void*),
uint, int32 lParam);
```

Übergibt *lParam* als **void*** und liefert einen **uint** zurück.

U_U_Dispatch

dispatch.h

```
int32 U_U_Dispatch(GENERIC& generic, uint (GENERIC::*pmf) (uint), uint,
int32 lParam);
```

Übergibt *lParam* als **uint** und liefert einen **uint** zurück.

U_U_U_U_Dispatch

dispatch.h

```
int32 U_U_U_U_Dispatch(GENERIC& generic, uint (GENERIC::*pmf) (uint,
uint, uint), uint wParam, int32 lParam);
```

Übergibt drei **uints** und liefert einen **uint** zurück Der erste **uint** ist *wParam,* der zweite **uint** ist das LOWORD von *lParam,* der dritte **uint** das HIWORD von *lParam.*

U_WPARAM_LPARAM_Dispatch

dispatch.h

```
int32 U_WPARAM_LPARAM_Dispatch(GENERIC& generic, uint (GENERIC::*pmf)
(uint, int32), uint wParam, int32 lParam);
```

Übergibt *wParam* als **uint** und *lParam* als **int32**; liefert einen **uint** zurück.

v_Activate_Dispatch

dispatch.h

```
int32 v_Activate_Dispatch(GENERIC& generic, void (GENERIC::*pmf) (uint,
uint, uint), uint wParam, int32 lParam);
```

Extraktionsmakro für WM_ACTIVATE-Botschaften. Übergibt drei **uints** und liefert immer 0 zurück. Unter Win32 ist der erste **uint** *wParam.lo,* der zweite **uint** ist *wParam.hi* und der dritte ist *lParam.* Unter Win16 ist der erste **uint** *wParam,.* der zweite **uint** *lParam.hi* und der dritte **uint** *lParam.lo.*

v_Dispatch

dispatch.h

```
int32 v_Dispatch (GENERIC& generic, void (GENERIC::*pmf) (), uint uint,
int32);
```

Übergibt *lparam* und liefert immer 0 zurück.

v_LPARAM_Dispatch

dispatch.h

```
int32 v_LPARAM_Dispatch (GENERIC& generic, void (GENERIC::*pmf) (int32),
uint wParam, int32 lParam);
```

Übergibt *lParam* als **32int** und liefert immer 0 zurück.

v_MdiActivate_Dispatch

dispatch.h

```
int32 v_MdiActivate_Dispatch(GENERIC& generic, void (GENERIC::*pmf)(uint,
uint), uint wParam, int32 lParam);
```

v_MdiActivate_Dispatch ist speziell für die Behandlung von *EvMDIActivate*-Botschaften ausgelegt und übergibt zwei **uints** sowie 0 als Rückgabewert.. Unter Win32 ist der erste **uint** *lParam* und der zweite *wParam*. Unter Win16 ist der erste **uint** *lParam.lo*. Wenn *lParam* ungleich Null ist, ist der zweite **uint** *lParam.hi*; andernfalls ist der zweite **uint** *wParam*.

v_ParentNotify_Dispatch

dispatch.h

```
int32 v_ParentNotify_Dispatch(GENERIC& generic, int(GENERIC::*pmf) (uint,
uint, uint), uint wParam, int32 lParam);
```

Führt Botschaftsextraktion für WM_PARENTNOTIFY-Botschaften aus, die ein übergeordnetes Fenster benachrichtigen, daß in einem ihrer untergeordneten Fenster ein angegebenes Ereignis stattgefunden hat. Übergibt drei **uints** und liefert immer 0 zurück. Für Win16 ist der erste **uint** *wParam*, der zweite *lParam.lo* und der dritte ist *lParam.hi*. Für Win32 ist der erste **uint** *wparam*, der zweite **uint** *lParam.lo* und der dritte **uint** *lParam.hi*, wenn *wparam.lo* eine Mausbotschaft ist. Andernfalls ist der erste **uint** *wparam.lo*, der zweite *lParam*, und der dritte **uint** ist *wParam.hi*.

v_POINT_Dispatch

dispatch.h

```
int32 v_POINT_Dispatch (GENERIC& generic, void (GENERIC::*pmf) (TPoint&),
uint wParam, int32 lParam);
```

Übergibt eine Referenz auf einen *POINT(TPoint&)* und liefert immer 0 zurück. Übergibt unter Win32 *lParam*, unter Win16 eine Referenz auf *lParam*.

v_POINTER_Dispatch

dispatch.h

```
int32 v_POINTER_Dispatch(GENERIC& generic, void (GENERIC::*pmf)(void*),
uint wParam, int32 lParam);
```

Übergibt einen *lParam* als **void***-Zeiger und liefert immer 0 zurück.

v_U_Dispatch

dispatch.h

```
int32 v_U_Dispatch(GENERIC& generic, void (GENERIC::*pmf) (uint, int32
lParam);
```

Übergibt *lParam* als **uint** und liefert immer 0 zurück.

v_U_POINT_Dispatch

dispatch.h

```
int32 v_U_POINT_Dispatch(GENERIC& generic, void (GENERIC::*pmf) (uint,
TPoint&), uint wParam, int32 lParam);
```

Übergibt *wParam* als einen **uint** und *lParam* als Referenz auf einen POINT; liefert immer 0 zurück.

v_U_U_Dispatch

dispatch.h

```
int32 v_U_U_Dispatch(GENERIC& generic, void (GENERIC::*pmf) (uint, uint),
uint wParam, int32 lParam);
```

Übergibt zwei **uints** und liefert immer 0 zurück. Der erste **uint** ist *wParam* und der zweite **uint** ist *lParam*.

v_U_U_U_Dispatch

dispatch.h

```
int32 v_U_U_U_Dispatch(GENERIC& generic, void (GENERIC::*pmf) (uint,
uint, uint), uint wParam, int32 lParam);
```

Übergibt drei **uints** und liefert immer 0 zurück. Der erste **uint** ist *wParam,* der zweite **uint** ist *lParam.lo* und der dritte **uint** ist *lParam.hi*.

v_U_U_W_Dispatch

dispatch.h

```
int32 v_U_U_W_Dispatch(GENERIC& generic, void (GENERIC::*pmf) (uint,
uint, uint), uint wParam, int32 lParam);
```

Übergibt drei **uints** und liefert immer 0 zurück. Unter Win32 ist der erste **uint** *wParam.lo,* der zweite ist *wParam.hi* und der dritte **uint** ist *lParam*. Unter Win16 ist der erste **uint** *wParam,* der zweite ist *lParam.lo* und der dritte **uint** ist *lParam.hi*. Dies ist einE teilweise benutzerdefiniertes Routine zum Extrahieren der Nachrichten WM_HSCROLL-, WM_VSCROLL- und WM_MENUSELECT.

v_WPARAM_Dispatch

dispatch.h

```
int32 v_WPARAM_Dispatch(GENERIC& generic, void (GENERIC::*pmf) (uint),
uint wParam, int32 lParam);
```

Übergibt *wParam* als **uint** und liefert immer 0 zurück.

v_WPARAM_LPARAM_Dispatch

dispatch.h

```
int32 v_WPARAM_LPARAM_Dispatch(GENERIC& generic, void (GENERIC::*pmf)
(uint, int32), uint wParam, int32 lParam);
```

Übergibt *wParam* als einen **uint** und *lParam* als einen **int32**; liefert immer 0 zurück.

K A P I T E L 5

Übersicht über ObjectComponents

ObjectComponents-Bibliotheken

In diesem Kapitel sind die Klassen, Makros, Konstanten, Datentypen und Registrierschlüssel von ObjectComponents alphabetisch verzeichnet. Neben jedem Eintrag ist die Header-Datei aufgeführt, die ihn definiert. Klassenelemente sind nach Zugriffsspezifizierern gruppiert **- public** oder **protected**. Innerhalb dieser Kategorien sind jeweils die Datenelemente, die Konstruktoren und Destruktoren sowie die Elementfunktionen zu Gruppen zusammengefaßt und alphabetisch aufgeführt.

Für jede Klasse werden nur die neuen oder neudefinierten Elemente aufgeführt. Von einer Basisklasse geerbte Elemente werden in der abgeleiteten Klasse nicht neuerlich verzeichnet. Private Datenelemente sind nicht aufgeführt. In einigen Fällen wurden auch öffentliche oder geschützte Datenelemente fortgelassen, nämlich dann, wenn sie nur für die Verwendung durch andere Klassen von ObjectComponents bestimmt sind.

Einige Einträge, die sich auf ObjectComponents-Programme beziehen, finden sich in anderen Abschnitten dieses Buches. OLE-fähige Klassen von ObjectWindows, wie etwa *TOleWindow* und *TOleFactory*, sind zusammen mit den anderen Klassen von ObjectWindows aufgeführt. Registriermakros und einige lokale Einträge sind weiter unten verzeichnet, in der Object Support Library (OSL)-Referenz.

Die folgenden Bibliotheken werden für das Linken von ObjectComponents-Anwendungen verwendet.

Medium	Large	DLL-Bibliotheken	Beschreibung
OCFWM.LIB	OCFWL.LIB	OCFWI.LIB	ObjectComponents
OWLWM.LIB	OWLWL.LIB	OWLWI.LIB	ObjectWindows
BIDSM.LIB	BIDSL.LIB	BIDSI.LIB	Klassenbibliotheken
OLE2W16.LIB	OLE2W16.LIB	OLE2W16.LIB	OLE System-DLLs
IMPORT.LIB	IMPORT.LIB	IMPORT.LIB	Windows System-DLLs

MATHWM.LIB	MATHWM.LIB		Fließkommaunterstützung
CWM.LIB	CWL.LIB	CRTLDLL.LIB	C-Laufzeitbibliotheken

ObjectComponents-Header-Dateien

Header-Dateien befinden sich im Unterverzeichnis INCLUDE/OCF Ihrer Festplatte und enthalten Deklarationen für Klassenfunktionen und Definitionen für Datentypen und Konstanten.

Datei	Inhalt
appdesc.h	*TAppDescriptor, TComponentFactory*
autodefs.h	Klassen und **structs**, die zur Automatisierung eingesetzt werden
automacr.h	Makros für Deklarationen und Definitionen von Automatisierungen
ocapp.h	OC_APPxxxx-Botschaften, *TOcApp, TOcFormatName, TOcHelp, TOcMenuDescr, TOcModule, TOcNameList, TOcRegistrar,* WM_OCEVENT-Botschaft
ocapp.rh	IDS_CFxxxx-Ressourcen-IDs für Zeichenketten, die Windows Standardformate für die Zwischenablage beschreiben
ocdefs.h	*TXObjComp*, HR_xxxx-Ergebniscodes, Deklarationsbezeichner
ocdoc.h	*TOcDocument*
ocfevx.h	Entschlüsselungsmakros für WM_OCEVENT-Botschaften
ocfpch.h	**#include**-Anweisungen für alle ObjectComponents-Header, zur Verwendung mit vorcompilierten Headern
ocobject.h	*TOcAspect, TOcDialogHelp, TOcDropAction, TOcInitHow, TOcInitInfo, TOcInitWhere, TOcInvalidate, TOcPartName, TOcScrollDir*
ocpart.h	*TOcPart, TOcPartCollection, TOcPartCollectionIter, TOcVerb*
ocreg.h	globale OCxxxx-Funktionen, ocrxxxx-Registrierkonstanten, *TAppMode* **enum**, *TRegistrar, TXRegistry*
ocremvie.h	*TOcRemView*
ocstorag.h	*TOcStream, TOcStorage*
ocview.h	OC_VIEWxxxx-Botschaften, *TOcDragDrop, TOcFormat, TOcFormatList, TOcFormatListIter, TOcSaveLoad, TOcScaleFactor,TOcToolBarInfo, TOcView, TOcViewPaint*
oleutil.h	DECLARE_COMBASES#-Makros, *TOleAllocator, TUnknown, TXOle*

Allgemeine Klassen, Makros und Typdefinitionen für OLE

Für die Erstellung von OLE-Anwendungen, gleich, ob sie Verknüpfen und Einbetten oder Automatisierung unterstützen, stellt ObjectComponents folgende Zusatztelemente zur Verfügung.

Element	Bedeutung
HR_xxxx-Makros	Rückgabewerte von OLE-Funktionen
_ICLASS-Makro	Bezeichner zum Deklarieren einer Klasse, die eine OLE-Schnittstelle implementiert
_IFUNC-Makro	Bezeichner für die Deklaration von OLE-Funktionen
_OCFxxxx-Makros	Bezeichner für die Deklaration von ObjectComponents-Klassen
Typdefinition TComponentFactory	Callback-Funktion, bei der eine Anwendung von OLE angeforderte Objekte erzeugt
Typdefinition TLocaleId	Datentyp der Bezeichner für die Ländereinstellung
Klasse TOleAllocator	Richtet einen Speicherzuweiser für die Verwendung durch OLE ein
Klasse TUnknown	Implementiert die grundlegende Schnittstelle *IUnknown*, die für alle OLE-Objekte erforderlich ist

Globale Zusatzfunktionen

Die Elemente in dieser Tabelle stellen Zusatzfunktionen dar, die ObjectComponents global deklariert.

Element	Zweck
DynamicCast	Wandelt einen Zeiger aus einem Typ in einen anderen um, sofern beide Typen durch Vererbung aufeinander bezogen sind
MostDerived	Liefert einen Zeiger auf den zum angegebenen Objekt passenden Typ der am weitesten abgeleiteten Klasse

ObjectComponents-Exception-Klassen

ObjectComponents löst Exceptions der in dieser Liste verzeichneten Typen aus. Alle Exception-Klassen sind aus TXBase abgeleitet.

Klasse	Zweck
TXAuto	Exceptions, die während der Automatisierung auftreten
TXObjComp	Exceptions, die bei Verknüpfungs- und Einbettungsoperationen von ObjectComponents auftreten
TXOle	Exceptions, die bei der Verarbeitung von OLE API-Befehlen auftreten
TXRegistry	Exceptions, die beim Zugriff auf die Registrierungsdatenbank des Systems auftreten

Automatisierungsklassen

ObjectComponents stellt die folgenden Klassen zur Verfügung, die Automatisierung unterstützen.

Klasse	Zweck
TAutoBase	Basisklasse zur Ableitung automatisierter Objekte
TAutoCommand	Enthält alle Daten für einen Befehl, der von einem Automatisierungs-Server empfangen wurde
TAutoEnumerator	Läßt einen Automatisierungs-Kontroller die Elemente in der Kollektion eines Servers aufzählen
TAutoIterator	Erlaubt es einem Automatisierungs-Server, die Elemente in einer automatisierten Kollektion zu iterieren
TAutoObject	Erzeugt einen smart-Zeiger auf ein automatisiertes Objekt
TAutoObjectByVal	Erlaubt einem Automatisierungs-Server die Automatisierung einer Methode, die ein Objekt über den Wert (statt über Referenz) zurückgibt
TAutoObjectDelete	Erlaubt einem Automatisierungs-Server die Automatisierung einer Methode, die ein Objekt zurückliefert
TAutoProxy	Basisklasse für die Ableitung der C++ Objekte, die ein Automatisierungskontroller zur Darstellung der OLE-Objekte, die er kontrollieren will, erzeugt
TAutoStack	Enthält eine Menge von *TAutoCommand*-Objekten, die jeweils einen von einem Automatisierungs-Server empfangenen Befehl repräsentieren
TAutoVal	Enthält die Daten einer VARIANT-Varianten; VARIANT ist das von OLE zum Senden von Automatisierungsbefehlen verwendete Datenformat.
TRegistrar	Verwaltet die Systemregistrierungsaufgaben für eine Automatisierungs-Anwendung

Aufzählungstypen und Typdefinitionen für Automatisierung

ObjectComponents stellt die folgenden Elemente zur Verfügung, die Automatisierung unterstützen.

Element	Zweck
AutoDataType enum	Bezeichner für Automatisierungsdatentypen
ObjectPtr typedef	Zeiger vom Typ **void** auf ein C++ Objekt

Automatisierungsdatentypen

In ObjectComponents stehen die folgenden Datentypen, die Automatisierung unterstützen, zur Verfügung. Näheres zur Verwendung dieser Datentypen unter Deklarationen und Definitionen.

Datentyp
TAutoBool struct
TAutoCurrency struct
TAutoDate struct
TAutoDouble struct
TAutoFloat struct
TAutoInt
TAutoLong struct
TAutoShort struct
TAutoString struct
TAutoStruct struct
TAutoVoid struct

Deklarationen/Definitionen von Automatisierungsdatentypen ocf/autodefs.h

Ein Automatisierungsdatentyp ist eine Struktur, deren einziger Zweck es ist, einen einzelnen Datentyp zu beschreiben. Automatisierungsdefinitionen verwenden diese Strukturen, um bei der Umwandlung von Parametern und Rückgabewerten zwischen den VARIANT-Varianten, die von OLE verwendet werden, und den C++ Datentypen, die Ihre Programme verwenden, mitzuhelfen.

Obwohl es sich um Strukturen handelt, lassen sich für die meisten von ihnen keine Instanzen erzeugen, da sie nicht über Konstruktoren verfügen und nur ein einziges statisches Datenelement enthalten. Sie sind alle von TAutoType abgeleitet und erben dessen *GetType*-Methode. Für die meisten dieser Strukturen besteht die einzige Aktion darin, auf *GetType*-Aufrufe durch Rückgabe der statischen ID für einen Datentyp zu antworten.

Die folgende Liste verzeichnet C++ Datentypen, die in Ihren Programmen auftauchen können, zusammen mit den entsprechenden Datentypen, die Sie in Automatisierungsdeklarationen (DECLARE_AUTOCLASS) und -definitionen (DEFINE_AUTOCLASS) verwenden sollten.

C++ Typ	Deklarationstyp	Definitionstyp
short	short	TAutoShort
unsigned short	short oder unsigned	TAutoShort oder TAutoLong
long	long	TAutoShort
unsigned long	unsigned long	TAutoLong (wird wie signed long behandelt)
int	int	TAutoInt
unsigned int	int oder long	TAutoShort oder TAutoLong
float	float	TAutoFloat

double	double	TAutoDouble
bool (oder int)	TBool	TAutoBool
TAutoDate	TAutoDate	TAutoDate
TAutoDate far*	TAutoDate far*	TAutoDateRef
TAutoCurrency far*	TAutoCurrency far*	TAutoCurrencyRef
char*	TAutoString	TAutoString
const char*	TAutoString	TAutoString
char far*	TAutoString	TAutoString
const char far*	TAutoString	TAutoString
string	string	TAutoString
enum	short oder int	TAutoShort oder benutzerdefiniertes AUTOENUM
T*	TAutoObject<T>	T (die Klasse T muß automatisiert sein)
T&	TAutoObject<T>	T (die Klasse T muß automatisiert sein)
const T*	TAutoObject<const T>	T (die Klasse T muß automatisiert sein)
const T&	TAutoObject<const T>	T (die Klasse T muß automatisiert sein)
T* (zurückgeliefert)	TAutoObjectDelete<T>	(C++ Objekt gelöscht, wenn keine Referenzen)
T& (zurückgeliefert)	TAutoObjectDelete<T>	(C++ Objekt gelöscht, wenn keine Referenzen)
T (zurückgeliefert)	TAutoObjectByVal<T>	T (T kopiert, gelöscht, wenn Referenzen==0)
void (keine Rückgabe)	(AUTOFUNCxV-Makros verwenden)	TAutoVoid
short far*	short far*	TAutoShortRef
long far*	long far*	TAutoLongRef
float far*	float far*	TAutoFloatRef
double far*	double far*	TAutoDoubleRef

Automatisierungsdeklarationsmakros ocf/automacr.h

Um OLE den Zugriff auf Teile einer automatisierten Klasse zu ermöglichen, erweitert ein Automatisierungs-Server die Deklaration der C++ Klasse um Deklarationsmakros und ihre Implementation um Definitionsmakros. Die Deklarationsmakros erzeugen Befehlsobjekte zur Ausführung der vom Kontroller gesendeten Befehle.

Der Block der Automatisierungsdeklarationsmakros beginnt immer mit DECLARE_AUTOCLASS. Es ist kein entsprechendes END-Makro zum Schließen der Deklaration erforderlich.

Makro	Bedeutung
DECLARE_AUTOCLASS(cls)	Die hierauf folgenden Makros deklarieren automatisierbare Elemente der benutzerdefinierten Klasse *cls*.

Deklarationsmakros

Nach DECLARE_AUTOCLASS folgt eine Serie vom Makros, eines für jedes Klassenelement, das Sie zur Verfügung stellen wollen. Welche Makros Sie im Einzelnen wählen, hängt von den Elementen ab.

Deklarationsmakro	Element
AUTODATA	Data
AUTOFLAG	Ein Bitflag
AUTOFUNC	Funktion
AUTOITERATOR	Iterator-Objekt
AUTOPROP	Eigenschaft
AUTOSTAT	Statisches Element oder globale Funktion
AUTOTHIS	***this**

Makro AUTODETACH

Eine Automatisierungsdeklaration kann zusätzlich noch das Makro AUTODETACH enthalten. Dieses Makro stellt kein Klassenelement zur Verfügung. Es hebt beim Entfernen des Objekts die Gültigkeit externer Referenzen auf.

Automatisierungsdefinitionsmakros ocf/automacr.h

Um OLE den Zugriff auf Teile einer automatisierten Klasse zu ermöglichen, erweitert ein Automatisierungs-Server die Deklaration der C++ Klasse um Deklarationsmakros und ihre Implementation um Definitionsmakros.

Der Block der Automatisierungsdefinitionsmakros beginnt immer mit DEFINE_AUTOCLASS und endet mit END_AUTOCLASS, es sei denn, das Objekt ist entweder selbst ein Component Object Model (COM)-Objekt, oder es hat von einem solchen Eigenschaften geerbt oder delegiert Aufgaben an ein solches. In diesem Fall beginnt der Block der Automatisierungsdefinitionsmakros mit DEFINE_AUTOAGGREGATE und endet mit END_AUTOAGGREGATE.

Makro	Bedeutung
DEFINE_AUTOCLASS(cls) END_AUTOCLASS(cls, name, doc, help)	Die hierauf folgenden Makros definieren automatisierbare Elemente der benutzerdefinierten Klasse *cls*. Die C++ Klasse *cls* wird OLE-Kontrollern unter dem Namen *name* zugänglich gemacht. Wenn der Anwender den Objektnamen über OLE erfragt, liefert das System

	die Zeichenkette in *doc* zurück. Wenn für das Objekt eine .HLP-Datei registriert ist, zeigt die Kontext-ID in *help* auf einen Hilfebildschirm, der das Objekt beschreibt.
DEFINE_AUTOAGGREGATE(cls, aggregator)	Die hierauf folgenden Makros definieren automatisierbare Elemente der benutzerdefinierten Klasse *cls*, die ein COM-Objekt darstellt, von einem solchen erbt oder an ein solches delegiert.
END_AUTOAGGREGATE(cls, name, doc, help)	Gleiche Funktion wie END_AUTOCLASS.

Zwischen den Makros DEFINE und END steht eine Serie anderer Makros, die jedes zugänglich gemachte Datenelement bzw. jede zugänglich gemachte Funktion beschreiben. Diese Makros implementieren Methoden für eine eingebettete Klasse innerhalb der automatisierten Klasse. Wenn ObjectComponents Befehle von einem Kontroller erhält, gibt es sie an diese eingebettete Klasse weiter. Die Makros erzeugen Rahmenfunktionen in der eingebetteten Klasse, die dieser das direkte Aufrufen der automatisierten Klasse ermöglichen.

Die Auswahl der Makros hängt im Einzelnen davon ab, welche Elemente zur Verfügung gestellt werden sollen.

Element	Deklarationsmakro
Automatisierte Anwendung	Makro EXPOSE_APPLICATION
Hilfsklasse	Makro EXPOSE_DELEGATE
Basisklasse	Makro EXPOSE_INHERIT
Kollektionsiterator	Makro EXPOSE_ITERATOR
Methode	Makros EXPOSE_METHOD
Eigenschaft Nur-Lesezugriff	EXPOSE_PROPRO
Eigenschaft Lese- und Schreibzugriff	EXPOSE_PROPRW
Eigenschaft Nur-Schreibzugriff	EXPOSE_PROPWO
Shutdown-Methode	EXPOSE_QUIT

Hook-Makros für die Automatisierung ocf/automacr.h

Diese Makros richten Hook-Prozeduren ein, die immer dann inkraft gesetzt werden, wenn ein bestimmter Automatisierungsbefehl ausgeführt wird. Sie werden niemals allein verwendet, sondern immer als letzter Parameter eines anderen Automatisierungsdeklarationsmakros. Wenn Sie ein solches Hook-Makro in die Deklaration eines zur Verfügung gestellten Klassenelements aufnehmen, dann führt ObjectComponents bei jedem Versuch eines Automatisierungskontrollers, diesen Befehl auszuführen, zunächst den Programmcode in der Hook-Prozedur aus. Dieser Code kann ein einfacher Ausdruck sein oder Aufrufe anderer Funktionen enthalten.

Die meisten der Makros erwarten als Parameter einen Ausdruck oder Programmcode. Häufig muß sich der Code oder der Ausdruck im Makro auf den Wert eines automatisierten Datenelements beziehen oder auf die Argumente, die in ihm übergeben werden. Innerhalb des Makro-Ausdrucks stehen *Arg1, Arg2, Arg3...* für die entgegengenommenen Argumente. *Val* steht für den Bezug auf ein automatisiertes Datenelement.

Makro	Bedeutung
AUTONOHOOK	Dieses Makro wird ohne Argumente verwendet und verhindert das Einschalten einer Hook-Prozedur in den Befehl. Der Aufruf kann dann nicht einmal von ObjectComponents überwacht werden. (Nur für sehr hochentwickelte Anwendungen).
AUTOINVOKE(code)	Hier eingesetzter Code wird bei jeder Ausführung des Automatisierungsbefehls ausgeführt. Eine AUTOINVOKE-Hook-Prozedur wird verwendet, wenn die normale Ausführungssequenz außer Kraft gesetzt werden soll.
AUTORECORD(code)	Hier eingesetzter Code erzeugt eine Aufzeichnung der ausgeführten Befehle, so daß die Sequenz gespeichert und wiedergegeben werden kann.
AUTOREPORT(code)	Der hier eingesetzte Code gibt einen Fehlercode vom automatisierten Element zurück. Wenn sich *code* zu 0 ergibt, geht OLE von einer erfolgreichen Ausführung des Befehls aus. Nimmt *code* einen Wert ungleich 0 an, löst OLE eine Exception aus, die einen Fehlercode an den Kontroller zurückliefert. Innerhalb des Code-Ausdrucks kann *Val* zum Bezug auf den tatsächlichen Rückgabewert verwendet werden.
AUTOUNDO(code)	Der hier eingesetzte Code erzeugt ein *TAutoCommand*-Objekt, das die Aktion des aktuellen Befehls rückgängig macht.
AUTOVALIDATE(condition)	Der hier eingesetzte Code sollte sich zu **true** berechnen, wenn die entgegengenommenen Argumente für den Befehl gültig sind, und ansonsten zu **false.** Wenn der Ausdruck **false** zurückliefert, löst OLE eine Exception aus, die einen Fehlercode an die Kontrolleranwendung zurückgibt.

Beispiel

Diese Deklaration stellt sicher, daß einem automatisierten Datenelement niemals ein Wert außerhalb eines angegebenen Bereichs zugewiesen wird.

```
AUTODATA(Zahl, Zahl, short,
 AUTOVALIDATE(Val=NUM_MIN && NotTooBig(Val)));
```

Platzhalter-Makros für die Automatisierung ocf/automacr.h

Um ein automatisiertes OLE-Objekt zu repräsentieren, erzeugt ein Automatisierungskontroller ein Platzhalterobjekt (aus *TAutoProxy* abgeleitet). Für jeden Befehl, den der Kontroller an das Objekt senden möchte, fügt er dem Platzhalterobjekt eine Methode hinzu. Die Platzhaltermethoden ahmen die vom Objekt unterstützten Befehle nach. Wenn der Kontroller Platzhaltermethoden aufruft, verschickt ObjectComponents durch OLE Automatisierungsbefehle.

Die Implementierung einer Platzhaltermethode enthält immer drei Makros: ein Makro AUTONAMES, ein Makro AUTOARGS und ein Makro AUTOCALL. AUTONAMES verknüpft Namen mit Argumenten. AUTOARGS beschreibt alle Argumente, die keinen Namen haben. Die Verwendung von Namen ermöglicht es, Teilmengen von Argumenten zu versenden und erlaubt es dem Server, den verbleibenden Argumenten Standardwerte zuzuweisen.

Das dritte Makro, AUTOCALL, informiert darüber, ob der Befehl eine Methode oder eine Eigenschaft des automatisierten Objekts repräsentiert, und ob er einen Wert zurückliefert.

Mit dem Dienstprogramm TYPEREAD.EXE (das sich im Unterverzeichnis OCTOOLS befindet) können Deklarationen und Definitionen für Platzhalterobjekte direkt erzeugt werden. TYPEREAD durchsucht die Typbibliothek eines Automatisierungs-Servers und erzeugt vollständigen Platzhaltercode für die Server-Steuerung.

Für die Behandlung von Sonderfällen können Sie ohne weiteres den Standard-Makrocode durch eigenen Code ersetzen.

Makro	Beschreibung
AUTONAMES	Verknüpft Namen mit Argumenten, so daß es dem Aufrufer freisteht, nur ausgewählte Argumente zu übergeben
AUTOARGS	Beschreibt Argumente, die nicht über Namen verfügen
AUTOCALL	Informiert darüber, ob der Befehl eine Methode oder eine Eigenschaft ist, und ob er einen Wert zurückliefert

Registrierschlüssel

Die meisten ObjectComponents-Programme erzeugen Registriertabellen, die ihre OLE-Fähigkeiten beschreiben. (Nur Automatisierungskontroller können diesen Schritt auslassen). Die Registriertabellen enthalten Paare von Schlüsseln und Werten. Die Schlüssel sind standardisiert. Sie entscheiden, welche Sie registrieren möchten, und stellen passende Werte zur Verfügung.

Je nachdem, ob es sich bei der Anwendung um einen Server, einen Container oder um ein Automatisierungsprogramm handelt, sind unterschiedliche Schlüssel auszuwählen. Einige Schlüssel müssen registriert werden, einige sind optional. Ferner beziehen sich einige nur auf die Stammregistriertabelle einer Anwendung, andere nur auf die Tabellen für jeden der Dokumenttypen der Anwendung.

Eine Registriertabelle beginnt mit dem Makro BEGIN_REGISTRATION und endet mit END_REGISTRATION. Zwischen diesen beiden befindet sich ein Makro für jeden zu registrierenden Schlüssel. Welche Makros zur Anwendung kommen, hängt von den jeweiligen Schlüsseln ab. Die meisten Schlüssel verwenden das Makro REGDATA, aber es gibt noch andere, wie etwa REGFORMAT und REGSTATUS.

Wenn die Anwendung ein Server ist, sind die meisten Informationen in seiner Registriertabelle auch in der Registrierungsdatenbank des Systems verzeichnet. Auf diese Weise kann OLE Informationen über den Server erhalten, ohne die Anwendung tatsächlich in den Arbeitsspeicher laden zu müssen. Wenn, beispielsweise, ein Automatisierungskontroller Informationen über die Befehle benötigt, die ein bestimmter Automatisierungs-Server unterstützt, kann OLE über den Eintrag in der Datenbank die Typbibliothek des Servers auffinden.

Schlüssel	Bedeutung
aspectall	Options-Flags, die für alle Präsentationsaspekte gelten.
aspectcontent	Options-Flags für die Inhalts-View eines Objekts.
aspectdocprint	Options-Flags für die Druckbild-View eines Objekts.
aspecticon	Options-Flags für die Symbolansicht eines Objekts.
aspectthumbnail	Options-Flags für die Miniaturbild-View eines Objekts.
clsid	Eine GUID zur Identifizierung der Anwendung.
cmdline	Argumente, die OLE beim Start des Servers in der Kommandozeile ausgeben sollte.
debugclsid	Eine GUID, die die Debug-Version eines Servers identifiziert. Wird intern generiert und sollte niemals direkt festgelegt werden.
debugdesc	Eine Zeichenkette vom Typ **long**, die die Debug-Version eines Programms beschreibt.
debugger	Der Dateiname und die Kommandozeilenschalter für das Laden des Debuggers.
debugprogid	Eine Zeichenkette, die die Debug-Version eines Programms benennt. Das Definieren dieses Strings zwingt ObjectComponents, debuggende und nicht-debuggende Versionen zu registrieren.
description	Eine Zeichenkette, die die Anwendung beschreibt.
directory	Das Standardverzeichnis für das Durchsuchen von Dokumentdateien.
docfilter	Dateispezifikation für die Auflistung von Dateien, die von der Anwendung generiert werden.
docflags	Options-Flags für die Dokumente der Anwendung.
extension	Eine aus drei Buchstaben bestehende Dateinamenserweiterung für die vom Server angelegten Dateien.
filefmt	Name des Standard-Dateiformats.
format*n*	Ein Format der Zwischenablage, das von der Anwendung unterstützt wird. (Verwenden Sie REGFORMAT, um Zwischenablageformate zu registrieren.)
handler	Vollständige Pfadangabe einer Bibliothek, die die vom Server erzeugten Objekte zeichnen kann. Die Voreinstellung ist OLE2.DLL.
helpdir	Vollständiger Pfad für das Verzeichnis, in dem sich die Online-Hilfe der Typbibliothek befindet.
iconindex	Ein Index, der angibt, welche der Symbole in den Server-Ressourcen die vom Server generierten Objekttypen repräsentieren. (Verwenden Sie REGICON, um Symbole zu registrieren.)
insertable	Zeigt an, daß die Anwendung ihr Dokument für das Verknüpfen und Einbetten in Container-Dokumenten zur Verfügung stellt.
language	Aktuell aktive Locale-Kennung. (Wird während der Automatisierung intern gesetzt).
menuname	Kurzname für den Server, der im Menü eines Containers verwendet wird.

path	Pfad, den OLE zum Auffinden des Servers benutzt. Dieser Schlüssel wird intern während der Registrierung gesetzt.
permid	Zeichenkette, die den Namen der Anwendung ohne jegliche Versionsangabe enthält.
permname	Zeichenkette, die eine Beschreibung der Anwendung ohne jegliche Versionsangabe enthält.
progid	Zeichenkette, die die Anwendung eindeutig benennt.
typehelp	Name der Datei, die die Online-Hilfe zur Typbibliothek enthält.
usage	Zeigt an, ob der Server die gleichzeitige Ausführung mehrerer Clients mit lediglich einer Instanz der Anwendung unterstützt.
verb*n*	Zeichenkette, die den Namen für eine Aktion enthält, die der Server mit seinen Objekten ausführen kann.
verb*n*opt	Options-Flags, die die Aktionsarten des Servers beschreiben. (Verwenden Sie REGVERBOPT, um die Optionen für Aktionsarten zu registrieren.)
version	Versions-String für die Anwendung und die Typbibliothek.

Klassen für Verknüpfung und Einbettung

Anwendungen, die Verknüpfen und Einbetten (Linking and Embedding) unterstützen, stellt ObjectComponents folgende Klassen zur Verfügung.

Klasse	Zweck
TOcApp	Verbinderobjekt, das BOCOLE-Schnittstellen für die Anwendung implementiert.
TOcDocument	Verwaltet die Teile im Verbunddokument eines Containers.
TOcFormatList	Liste der Datenformate für die Zwischenablage, die von einem Dokument unterstützt werden.
TOcFormatListIter	Iterator für diese Liste der Datenformate für die Zwischenablage, die von einem Dokument unterstützt werden.
TOcFormatName	Enthält Strings, die ein einzelnes Datenformat beschreiben, auf das eine Anwendung in der Zwischenablage treffen kann (siehe *TOcNameList*).
TOcModule	Basisklasse für die Ableitung OLE-fähiger Anwendungsobjekte.
TOcNameList	Enthält eine Kollektion von Strings, die alle Datenformate beschreiben, auf die eine Anwendung in der Zwischenablage treffen kann.
TOcPart	Verbinderobjekt, das von einem Container verwendet wird, um ein verknüpftes oder eingebettetes Objekt in einem seiner Dokumente zu repräsentieren.
TOcPartCollection	Verwaltet eine Kollektion verknüpfter oder eingebetteter Teile.
TOcPartCollectionIter	Iterator für die Kollektion von Teilen, die in einem einzelnen Dokument eingebunden oder mit ihm verknüpft sind.
TOcRegistrar	Verwaltet die OLE-Registrieraufgaben für eine Anwendung, die Verknüpfen und Einbinden unterstützt.
TOcRemView	Verbinderobjekt, das ein Server zum Zeichnen eines Objekts verwendet, das im Dokument eines Containers eingebettet oder mit ihm verknüpft ist.
TOcScaleFactor	Übermittelt Informationen von einem Container an einen Server, mit denen der Server aufgefordert wird, verknüpfte oder eingebettete Objekte in einem bestimmten Maßstab zu zeichnen.
TOcVerb	Enthält Informationen über eine Aktionsart, die ein Server mit seinen eigenen Objekten ausführen kann, wenn sie in einem Container eingebettet oder mit ihm verknüpft sind.
TOcView	Ein Verbinderobjekt, das eine Anwendung verwendet, um ihre eigenen Dokumente in ihrem eigenen Rahmenfenster zu zeichnen.

Aufzählungstypen für Verknüpfung und Einbettung

ObjectComponents stellt Anwendungen, die Verknüpfen und Einbetten (Linking and Embedding) unterstützen, folgende Aufzählungstypen zur Verfügung.

Element	Beschreibung
TOcAppMode enum	Flags zur Identifizierung der Ausführungsbedingugen der Anwendung
TOcAspect enum	Flags zur Identifizierung der Aspekte der Objektpräsentation
TOcDialogHelp enum	Konstanten, die OLE Standard-Dialogfenster identifizieren, in denen ein Anwender Hilfe anfordern kann.
TOcDropAction enum	Konstanten, die Aktionen identifizieren, die sich aus dem Ablegen eines Objekts in einem Fenster ergeben können.
TOcInitHow enum	Konstanten, die die Aktion identifizieren, die ein Container beim Entgegennehmen eines neuen Objekts ausführen muß - Einbetten oder Verknüpfen.
TOcInitWhere enum	Konstanten, die die Orte identifizieren, an denen sich die Daten für ein Objekt befinden können.
TOcInvalidate enum	Flags, die anzeigen, ob ein Objekt aufgrund einer Änderung seiner Daten oder lediglich aufgrund einer Änderung seiner Erscheinung ungültig ist.
TOcPartName enum	Diese Konstanten identifizieren verschiedene Strings, die von einem Container angefordert werden könnten, der nach dem Namen eines in ihm eingebetteten oder mit ihm verknüpften Objekts sucht.
TOcScrollDir enum	Konstanten, die die Richtungen für einen eventuellen Bildlauf angeben, der beim Container angefragt werden könnte.

Botschaften für Verknüpfung und Einbettung

Anwendungen, die Verknüpfen und Einbetten (Linking and Embedding) unterstützen, stellt ObjectComponents folgende Botschaften zur Verfügung.

Botschaft	Zweck
OC_APPxxxx	Botschaften, die an ein Anwendungsobjekt geschickt werden
OC_VIEWxxxx	Botschaften, die an ein View-Objekt geschickt werden
WM_OCEVENT	Transportiert Ereignissignale von ObjectComponents an eine Anwendung.

Strukturen für Verknüpfung und Einbettung

Anwendungen, die Verknüpfen und Einbetten (Linking and Embedding) unterstützen, stellt ObjectComponents folgende Strukturen zur Verfügung.

TOcDragDrop struct	Enthält Informationen, die ein Container benötigt, damit er ein in seinem Fenster abgelegtes Objekt entgegennehmen kann.
TOcInitInfo struct	Enthält Informationen, die ein Container benötigt, um ein neues Objekt in seinem Dokument zu plazieren.
TOcMenuDescr struct	Enthält Informationen über ein gemeinsam genutztes Menü, in das Container und Server ihre Befehle gemeinsam einbringen, um eine direkte Bearbeitung im Container-Dokument zu ermöglichen.
TOcSaveLoad struct	Enthält Informationen, die eine Anwendung zum Laden oder Speichern eines verknüpften oder eingebetteten Objekts benötigt.
TOcToolBarInfo struct	Enthält Handles für die Mauspalette eines Servers, die während der direkten Bearbeitung im Container-Dokument im Fenster des Containers angezeigt werden soll.
TOcViewPaint struct	Enthält Informationen für den Server, wie ein verknüpftes oder eingebettetes Objekt neu gezeichnet werden soll, wenn der Container einen Teil der Objektoberfläche für ungültig erklärt hat.

ocrxxxx-Konstanten ocf/ocreg.h

Der Header ocreg.h definiert eine Anzahl von Konstanten, die beim Aufbau der Registriertabellen einer Anwendung verwendet werden. Diese Konstanten beginnen alle mit *ocr*. Es lassen sich mehrere Gruppen unterscheiden. Die meisten der Konstanten werden zusammen mit dem Makro REGFORMAT verwendet und beschreiben dann die verschiedenen Arten des Datenaustauschs, die ein Dokument unterstützt.

Gruppe	Bedeutung
Aspekt-Konstanten	Arten der Datendarstellung (wie Symbol, Inhalt oder Miniaturbild)
Zwischenablage-Konstanten	Datenformate der Zwischenablage (wie Text, Bitmap oder Verknüpfungsquelle)
Richtungskonstanten	Datenübertragungsrichtung (Empfangen oder Senden)
Begrenzungskonstante	Höchstzahl der Elemente, die registriert werden können
Medienkonstanten	Medien für den Datenaustausch (wie Datei oder Zwischenablage)
Objektstatus-Konstanten	Darstellungsoptionen (wie etwa auf Symbol verkleinern oder bei Größenänderung neu zeichnen)
Verwendungskonstanten	Unterstützung mehrerer Clients (einfache oder mehrfache Verwendung)
Attributkonstanten Aktionsart	Options-Flags Aktionsart (nie dirty und im Menü auflisten)
Menü-Flags Aktionsart	Anzeigeoptionen Aktionsart (wie grau dargestellt, deaktiviert oder Menüleiste)

K A P I T E L 6

Bibliotheksreferenz ObjectComponents

Makro _ICLASS ocf/oleutil.h

Modifiziert die Deklaration einer Schnittstellenklasse, die eine Schnittstelle für OLE oder für die BOCOLE Support-Bibliothek definiert oder implementiert.

Makro _IFUNC ocf/oleutil.h

Modifiziert die Deklaration einer OLE-Funktion.

Das Makro _IFUNC steuert die Aufrufkonventionen und Exportdeklarationen von Funktionen. Wenn diese Makros in einem Schlüsselwort eingesetzt werden, kann der Compiler die richtige Kombination von Modifizierern für eine bestimmte Umgebung auswählen.

ObjectComponents verwendet das Makro sowohl für die Deklaration von OLE- und BOCOLE-Funktionen als auch für Elementfunktionen, die als Rahmenfunktionen für direkte OLE- und BOCOLE-Aufrufe dienen.

_IFUNC dient in Borland-Headern den gleichen Zwecken wie STDMETHODCALLTYPE in OLE-System-Headern.

Makros _OCFxxxx ocf/ocfdefs.h

Diese Makros werden intern für die Deklaration von Klassen, Funktionen und Datenelementen in ObjectComponent-Klassen verwendet. Je nachdem, ob sie für eine 16- oder 32-Bit .EXE- oder eine .DLL-Datei eingesetzt

werden, unterscheidet sich ihre Definition. Einige der Makros erzwingen außerdem die Deklaration zu **_huge**.

Diese Makros stimmen weitgehend mit den entsprechenden *_OWLxxxx*-Makros überein.

Konstante	Bedeutung
_OCFCLASS	Exportiert oder importiert Klassen für DLLs.
_OCFDATA	Exportiert oder importiert Datenelemente für DLLs.
_OCFFUNC	Exportiert oder importiert Elementfunktionen für DLLs.

Registrierschlüssel aspectall

Registriert Options-Flags, die alle Views eines Objekt betreffen. Die Flags steuern die Art und Weise, in der die Views des Objekts dargestellt werden.

Verknüpfungs- und Einbettungs-Server können optional den Aspektstatus in ihren Dokument-Registriertabellen registrieren. Der Aspektstatus betrifft aber nicht Anwendungs-Registriertabellen, Container oder Automatisierungs-Server.

Um Flags für alle Aspektarten zu registrieren, wird das Makro REGSTATUS verwendet. In diesem Fall wird "all" als erster Parameter übergeben und ein Wert für den Aufzählungstyp *ocrxxxx*Object Status **enum** als zweiter Parameter.

```
REGSTATUS(all, ocrNoSpecialRendering)
```

Siehe auch: Registrierschlüssel aspectcontent, Registrierschlüssel aspectdocprint, Registrierschlüssel aspecticon, Registrierschlüssel aspectthumbnail, ocrxxxObject Status enum, Makro REGSTATUS.

Registrierschlüssel aspectcontent

Registriert Options-Flags für die Inhalts-View eines Objekts. Die Inhalts-View zeigt normalerweise alle Daten in einem Objekt an (oder so viele, wie in dem zur Verfügung stehenden Platz dargestellt werden können). Die Options-Flags bestimmen, auf welche Art und Weise die Inhalts-View verwendet wird.

Verknüpfungs- und Einbettungs-Server können optional den Aspektstatus in ihren Dokument-Registriertabellen registrieren. Der Aspektstatus

betrifft aber nicht Anwendungs-Registriertabellen, Container oder Automatisierungs-Server.

Um Flags für die Inhaltsdarstellung zu registrieren, wird das Makro REGSTATUS verwendet. "content" wird als erster Parameter übergeben und ein Wert für den Aufzählungstyp *ocrxxxx*Object Status **enum** als zweiter Parameter.

```
REGSTATUS(content, ocrRecomposeOnResize)
```

Siehe auch: Registrierschlüssel aspectall, Registrierschlüssel aspectdocprint, Registrierschlüssel aspecticon, Registrierschlüssel aspectthumbnail, ocrxxxObject Status enum, Makro REGSTATUS.

Registrierschlüssel aspectdocprint

Registriert Options-Flags, die die Druckbild-View eines Dokuments betreffen. Die Druckausgabe-View gibt normalerweise eine annähernde Darstellung des Objekts, so wie es auf dem aktuellen Drucker ausgegeben werden würde. Mit den Options-Flags wird festgelegt, wie die Druckausgabe-View angezeigt wird.

Verknüpfungs- und Einbettungs-Server können optional den Aspektstatus in ihren Dokument-Registriertabellen registrieren. Der Aspektstatusstatus betrifft aber nicht Anwendungs-Registriertabellen, Container oder Automatisierungs-Server.

Um Flags für die Druckausgabe-View zu registrieren, wird das Makro REGSTATUS verwendet. "docprint" wird als erster Parameter übergeben und ein Wert für den Aufzählungstyp *ocrxxxx*Object Status **enum** als zweiter Parameter.

Siehe auch: Registrierschlüssel aspectall, Registrierschlüssel aspectcontent, Registrierschlüssel aspecticon, Registrierschlüssel aspectthumbnail, ocrxxxObject Status enum, Makro REGSTATUS.

Registrierschlüssel aspecticon

Registriert Options-Flags, die die Symbol-View eines Objekts betreffen. Die Symbol-View gibt nicht den Inhalt eines Objektes preis, sondern zeigt vielmehr ein Symbol an, das für eine bestimmte Art von Objekt steht.

Verknüpfungs- und Einbettungs-Server können optional den Aspektstatus in ihren Dokument-Registriertabellen registrieren. Der Aspektstatus betrifft aber nicht Anwendungs-Registriertabellen, Container oder Automatisierungs-Server.

Um Flags für die Symboldarstellung zu registrieren, wird das Makro REGSTATUS verwendet. "icon" wird als erster Parameter übergeben und ein Wert für den Aufzählungstyp *ocrxxxx*Object Status **enum** als zweiter Parameter.

```
REGSTATUS(icon, ocrOnlyIconic)
```

Siehe auch: Registrierschlüssel aspectall, Registrierschlüssel aspectcontent, Registrierschlüssel aspectdocprint, Registrierschlüssel aspectthumbnail, ocrxxxObject Status enum, Makro REGSTATUS.

Registrierschlüssel aspectthumbnail

Registriert Options-Flags, die die Miniaturbild-View eines Objekts betreffen. Die Miniaturbild-View ist normalerweise eine stark verkleinerte Darstellung des Inhalts eines Objekts. Die Flags steuern die Art und Weise, wie die Miniaturbild-View angezeigt wird.

Verknüpfungs- und Einbettungs-Server können optional den Aspektstatus in ihren Dokument-Registriertabellen registrieren. Der Aspektstatus betrifft aber nicht Anwendungs-Registriertabellen, Container oder Automatisierungs-Server.

Um Flags für die Miniaturbild-Darstellung zu registrieren, wird das Makro REGSTATUS verwendet. "thumbnail" wird als erster Parameter übergeben und ein Wert für den Aufzählungstyp *ocrxxxx*Object Status **enum** als zweiter Parameter.

Siehe auch: Registrierschlüssel aspectall, Registrierschlüssel aspectcontent, Registrierschlüssel aspectdocprint, Registrierschlüssel aspecticon, ocrxxxObject Status enum, Makro REGSTATUS.

Makros AUTOARGS ocf/automacr.h

Ein Automatisierungskontroller verwendet AUTOARGS zur Implementierung von Methoden in seinen Platzhalterobjekten. AUTOARGS-Makros verzeichnen alle Argumente, die der Kontroller an einen Automatisierungsbefehl übergibt, in einer Liste und identifizieren sie anhand der willkürlichen Parameternamen aus der Funktionsdefinition.

AUTOARGS-Makros stellen den zweiten von drei Makrosätzen dar, die zur Implementierung von Methoden in Platzhalterobjekten verwendet werden. Der erste, AUTONAMES, vergibt Namen an beliebige Argumente, die der Kontroller über den Namen referenzieren möchte. Der dritte

Makrosatz, AUTOCALL, informiert darüber, ob ein Befehl eine Methode oder eine Eingenschaft ist, und ob er einen Rückgabewert hat.

Der Header automacr.h definiert AUTOARG-Makros, die bis zu sechs Argumente annehmen. Zur Generierung von Versionen, die mehr als sechs Argumente annehmen, wird das Dienstprogramm MACROGEN.EXE verwendet.

Makro	Bedeutung
AUTOARGS0()	Der Automatisierungsbefehl benötigt keine Argumente.
AUTOARGS1(a1)	Der Automatisierungsbefehl benötigt Argument *a1*.
AUTOARGS2(a1, a2)	Der Automatisierungsbefehl benötigt die Argumente *a1* und *a2*.
AUTOARGS3(a1, a2, a3)	Der Automatisierungsbefehl benötigt die Argumente *a1*, *a2* und *a3*.
AUTOARGS4(a1, a2, a3, a4)	Der Automatisierungsbefehl benötigt die Argumente *a1*, *a2*, *a3* und *a4*.

Makros AUTOCALL_xxxx ocf/automacr.h

AUTOCALL ist der dritte der drei Makrosätze, die ein Automatisierungskontroller zur Implementierung von Automatisierunsbefehlen in Platzhalterobjekten verwendet. Die beiden ersten Sätze, AUTONAMES und AUTOARGS beschreiben die Argumente des Befehls. AUTOCALL-Makros informieren darüber, ob der Befehl eine Methode oder eine Eigenschaft des automatisierten Objekts repräsentiert, und ob der Befehl einen Wert zurückliefert. Befehle, deren Rückgabewert seinerseits ein automatisiertes Objekt ist, müssen darüber hinaus besonders gekennzeichnet sein.

Makro	**Bedeutung**
AUTOCALL_METHOD_REF(prx)	Der Befehl ist eine Methode, die eine Referenz auf ein Objekt zurückliefert. *prx* ist ein aus *TAutoProxy* abgeleitetes Objekt, das den Rückgabewert empfängt.
AUTOCALL_METHOD_RET	Der Befehl ist eine Methode, die einen Wert zurückgibt.
AUTOCALL_METHOD_VOID	Der Befehl ist eine Methode, die keinen Wert zurückgibt.
AUTOCALL_PROP_GET	Der Befehl liefert den Wert einer Eigenschaft des automatisierten Objekts zurück.
AUTOCALL_PROP_REF(prx)	Der Befehl liefert den Wert einer Eigenschaft zurück, und dieser Wert ist seinerseits ein Objekt. *prx* ist ein aus *TAutoProxy* abgeleitetes Objekt, das den Rückgabewert empfängt.
AUTOCALL_PROP_SET(val)	Der Befehl weist *val* einer Eigenschaft des automatisierten Objekts zu.

Makro _AUTOCLASS ocf/autodefs.h

_AUTOCLASS ist der Klassenmodifizierer, den ObjectComponents zur Deklaration der Basisklassen und Elementobjekte verwendet, die es zum Zweck der Automatisierung innerhalb Ihrer Klassen erzeugt. Sofern Ihre Anwendung das Speichermodell der Anwendungsumgebung verwendet, brauchen Sie sich nicht weiter mit _AUTOCLASS zu befassen, das in Voreinstellung als leer definiert ist. Wenn Sie Ihre Automatisierungsklassen jedoch mit einem Modifizierer deklarieren, der vom Modell der umgebenden Klasse abweicht, müssen die von ObjectComponents definierten Klassen (wie etwa *TAutoBase*) entsprechend angepaßt werden. Die Modifikation nehmen Sie vor, indem Sie _AUTOCLASS selbst definieren. Zum Beispiel:

```
#define_AUTOCLASS__far
```

Makros AUTODATA ocf/automacr.h

Ein Automatisierungs-Server verwendet AUTODATA-Makros in einer Automatisierungsdeklaration (nach DECLARE_AUTOCLASS), um die Datenelemente einer automatisierten Klasse für den Zugriff durch OLE zugänglich zu machen.

Beide Formen akzeptieren die gleichen vier Parameter. *name* ist der interne Name, den Sie dem Datenelement zuweisen. ObjectComponents verwendet die internen Namen zur Verwaltung der automatisierten Elemente. Die einzige Stelle, an der dieser Name außerdem verwendet wird, ist die nachfolgende Automatisierungsdefinition (nach DEFINE_AUTOCLASS).

member ist der C++ Name des Datenelements, den Sie normalerweise im Quellcode verwenden.

In den meisten Fällen sollte *type* ein normaler C++ Datentyp sein. Wenn aber das Datenelement ein String oder ein Objekt ist, geben Sie hier stattdessen *TAutoString* oder eine der *TAutoObject*-Klassen an. Näheres dazu finden Sie unter Automatisierungsdatentypen.

options ist die Stelle, an der eine Hook-Prozedur eingeschaltet werden kann. Der Code einer Hook-Prozedur wird bei jeder Ausführung des Automatisierungsbefehls aufgerufen. Hook-Prozeduren können Befehle aufzeichnen, sie rückgängig machen oder validieren. Näheres finden Sie unter Hook-Makros für die Automatisierung. Der Parameter *options* kann weggelassen werden, doch muß dem vorangehenden Argument in jedem Fall ein Komma folgen.

Makro	Bedeutung
AUTODATA(name, member, type, options)	Der Befehl erlaubt Lese- und Schreibzugriff auf ein Datenelement.
AUTODATARO(name, member, type, options)	Der Befehl erlaubt lediglich Lesezugriff auf ein Datenelement.

Aufzählungstyp AutoDataType ocf/autodefs.h

enum AutoDataType

Diese Flags identifizieren Automatisierungsdatentypen. Die Typen entsprechen Standard OLE 2-Datentypen. Die Klasse *TAutoVal* verwendet die Flags als Leitfaden zur Konvertierung nach und aus den VARIANT-Vereinigungsmengen, die OLE zwischen Programmen austauscht.

Konstante	Bedeutung
atVoid	**void**
atNull	SQL-Stil Null
atShort	2-Byte **signed int**
atLong	4-Byte **signed int**
atFloat	4-Byte real
atDouble	8-Byte real
atCurrency	Währung
atDatetime	Zeit/Datum als **double**
atString	BSTR, String mit vorangestellter Länge
atObject	*IDispatch**
atError	SCODE
atBool	**True** = -1, **false** = 0
atVariant	VARIANT FAR*
atUnknown	*IUnknown**

Die vorstehenden Flags schließen sich gegenseitig aus. Ein Wert kann nur zu einem Typ gehören. Jedes der Typflags kann aber mit den folgenden Bitflags kombiniert werden.

Bitflag	Bedeutung
atTypeMask	Basis-Typcode ohne Bitflags.
atOLE2Mask	Typcode mit Bitflags.
atByRef	Der Wert ist eine Referenz auf ein Objekt.
atEnum	Der Wert ist eine Aufzählung bestimmten Typs.

Siehe auch: Klasse TAutoVal, Konstruktor TAutoEnumTpublic

Makro AUTODETACH ocf/automacr.h

AUTODETACH

Ein Automatisierungs-Server verwendet dieses Makro in seiner Automatisierungsdeklaration (nach DECLARE_AUTOCLASS) um sicherzustellen, daß OLE benachrichtigt wird, wenn das automatisierte Objekt entfernt wird. Die Weiterleitung der Todesanzeige an OLE verhindert Systemabstürze, die eintreten könnten, wenn ein Kontroller versucht, das nicht-existente Objekt zu manipulieren. Diese Benachrichtigung ist nur dann erforderlich, wenn die Logik Ihres Programms ein Entfernen des automatisierten Objekts auf nicht-automatischem Wege erlaubt, während es noch mit dem Kontroller verbunden ist.

Die Ableitung einer Klasse aus *TAutoBase* dient genau dem gleichen Zweck. Der Vorteil von AUTODETACH besteht darin, daß Sie es für die Automatisierung von Klassen verwenden können, die nicht von Ihnen erzeugt wurden und auf deren Ableitung Sie keinen Einfluß haben.

Makros AUTOENUM ocf/automacr.h

Ein Automatisierungs-Server verwendet die AUTOENUM-Makros, um Automatisierungskontrollern Aufzählungswerte zur Verfügung zu stellen. Wenn beispielsweise der Server den Kontroller veranlassen möchte, Aktionen in einen Befehl TueDies zu übergeben, könnte er zu diesem Zweck einen Aufzählungstyp erzeugen, der Werte wie Wiedergabe, Stop und Rückspulen enthält. Um dem Kontroller diese Werte zugänglich zu machen, muß der Server eine AUTOENUM-Tabelle anlegen. In diesem Beispiel besteht die Tabelle aus drei AUTOENUM-Makros. eines für jeden Aufzählungswert.

```
DEFINE_AUTOENUM(TAction, TAutoShort);
 AUTOENUM("Wiedergabe", Wiedergabe)
 AUTOENUM("Stop", Stop)
 AUTOENUM("Rückspulen", Rückspulen);
END_AUTOENUM(TAction, TAutoShort)
```

Makro	Bedeutung
DEFINE_AUTOENUM(cls, type)	Steht am Anfang einer AUTOENUM-Tabelle. *cls* ist der Name des automatisierten Aufzählungstyps (nicht der Name des C++ Aufzählungstyps). Diesen Namen können Sie willkürlich vergeben. Er wird sonst nur noch in der Automatisierungsdefinition der Anwendung verwendet. *type* ist der Automatisierungsdatentyp, der beschreibt, welche Art von Werten aufgezählt wird. Weitere Informationen siehe unter Automatisierungsdatentypen.
AUTOENUM(name, val)	*name* ist der öffentliche String, den ein Kontroller verwendet, um sich auf einen

	bestimmten aus einer Reihe von Aufzählungswerten zu beziehen. *val* ist der interne Wert, den der Server mit *name* verbindet.
END_AUTOENUM(cls, type)	Schließt eine AUTOENUM-Tabelle. *cls* und *type* sind die gleichen wie bei DEFINE_AUTOENUM.

Makro AUTOFLAG ocf/automacr.h

AUTOFLAG(name, data, mask, options)

Ein Automatisierungs-Server verwendet AUTOFLAG in einer Automatisierungsdeklaration (nach DECLARE_AUTOCLASS), um ein einzelnes Bit aus einer Menge von Bitflags für die Automatisierung verfügbar zu machen.

name ist ein interner Name, den Sie dem Bit geben. ObjectComponents verwendet interne Namen zur Verwaltung der automatisierten Elemente. Dieser Name wird darüber hinaus nur noch in der nachfolgenden Automatisierungsdefinition verwendet (nach DEFINE_AUTOCLASS).

data ist der C++ Name eines Datenelements, das eine Menge von Bitflags enthält.

mask ist ein Wert mit einem gesetzten Bit, das die Position des zur Verfügung gestellten Flags in *data* markiert.

In *options* kann eine Hook-Prozedur eingeschaltet werden. Der Code einer Hook-Prozedur wird bei jeder Ausführung des Automatisierungsbefehls aufgerufen. Hook-Prozeduren können Befehle aufzeichnen, sie rückgängig machen oder validieren. Näheres finden Sie unter Hook-Makros für die Automatisierung. Der Parameter *options* kann weggelassen werden, doch muß dem vorangehenden Argument in jedem Fall ein Komma folgen.

Makros AUTOFUNC ocf/automacr.h

Ein Automatisierungs-Server verwendet AUTOFUNC-Makros in einer Automatisierungsdeklaration (nach DECLARE_AUTOCLASS), um Elementfunktionen einer automatiserten Klasse für den Zugriff durch OLE zugänglich zu machen.

In jeder Version des Makros ist der erste Parameter, *name*, ein interner Name, den Sie der Funktion zuweisen. ObjectComponents verwendet die internen Namen zur Verwaltung der automatisierten Elemente. Dieser Name wird sonst nur noch in der nachfolgenden Automatisierungsdefinition verwendet (nach DEFINE_AUTOCLASS).

func ist der C++ Name der Eelementfunktion, der name, den Sie normalerweise in Ihrem Quellcode verwenden.

ret ist der Datentyp, den die Funktion zurückliefert. *type1*, *t1*, *t2*, *t3* und *t4* repräsentieren die Datentypen der Parameter. In den meisten Fällen sollten diese Datentypen normale C++ Datentypen sein. Wenn das Datenelement aber ein String oder ein Objekt ist, geben Sie hier stattdessen *TAutoString* oder eine der *TAutoObject*-Klassen an. Näheres finden Sie unter Automatisierungsdatentypen. Ferner können automatisierte Funktionen keine Werte vom Typ **const** zurückliefern. Verwenden Sie const also nicht in einem *ret*-Typ.

In *options* kann eine Hook-Prozedur eingeschaltet werden. Der Code einer Hook-Prozedur wird bei jeder Ausführung des Automatisierungsbefehls aufgerufen. Hook-Prozeduren können Befehle aufzeichnen, sie rückgängig machen oder validieren. Näheres finden Sie unter Hook-Makros für die Automatisierung. Der Parameter *options* kann weggelassen werden, doch muß dem vorangehenden Argument in jedem Fall ein Komma folgen.

Der Header automacr.h definiert Versionen dieses Makros, die bis zu drei Argumente annehmen. Zur Generierung von Versionen, die mehr als drei Argumente annehmen, wird das Dienstprogramm MACROGEN.EXE verwendet.

Makro	Bedeutung
AUTOFUNC0(name, func, ret, options)	Die Funktion nimmt keine Parameter an und liefert einen Wert vom Typ *ret* zurcük.
AUTOFUNC0V(name, func, options)	Die Funktion nimmt keine Parameter an liefert **void** zurück.
AUTOFUNC1(name, func, ret, type1, options)	Die Funktion nimmt einen Parameter vom Typ *type1* an und liefert einen Wert von Typ *ret* zurück.
AUTOFUN1V (name, func, type1, options)	Die Funktion nimmt einen Parameter vom Typ *type1* an und liefert **void** zurück.
AUTOFUNC2(name, func, ret, t1, t2, options)	Die Funktion akzeptiert zwei Parameter der Typen *t1* und *t2*. Sie gibt einen Wert vom Typ *ret* zurück.
AUTOFUNC2V(name, func, t1, t2, options)	Die Funktion akzeptiert zwei Parameter der Typen *t1* und *t2* und liefert void zurück.
AUTOFUNC3(name, func, ret, t1, t2, t3, options)	Die Funktion akzeptiert drei Parameter und liefert einen Wert zurück.
AUTOFUNC3V(name, func, t1, t2, t3, options)	Die Funktion akzeptiert drei Parameter und liefert **void** zurück.
AUTOFUNC4(name, func, ret, t1, t2, t3, t4, options)	Die Funktion akzeptiert vier Parameter und liefert einen Wert zurück.
AUTOFUNC4(name, func, t1, t2, t3, t4, options)	Die Funktion akzeptiert vier Parameter und liefert **void** zurück.

Makro AUTOINVOKE ocf/automacr.h

AUTOINVOKE(code)
Ein Automatisierungs-Server verwendet AUTOINVOKE in einem Automatisierungsdeklarations-Makro, um benutzerdefinierten Code einzuschalten, den ObjectComponents jedesmal ausführen soll, wenn die Anwendung einen bestimmten Automatisierungsbefehl erhält. *code* ist der Ausdruck oder Funktionsaufruf, der bei jedem Befehl ausgeführt werden soll.

Erzeugen Sie mit AUTOINVOKE eine Hook-Prozedur, wenn Sie die normale Ausführungssequenz außer Kraft setzen wollen.

Makros AUTOITERATOR ocf/automacr.h

Ein Iterator ist ein Objekt, das zum Aufzählen einer Kollektion von Objekten verwendet wird. Die Methoden eines Iterators führen einen Aufrufer Schritt für Schritt durch eine Liste von Objekten und lassen ihn jedes einzelne der Reihe nach inspizieren.

Ein Automatisierungs-Server muß in jedem automatisierten Objekt, das eine Kollektion anderer Objekte repräsentiert, einen Iterator erzeugen. Zum Erzeugen eines Iterators fügt der Server der Automatisierungsdefinition der Klasse eines der AUTOITERATOR-Makros hinzu (nach DEFINE_AUTOCLASS). Der Iterator muß außerdem mit dem Makro EXPOSE_ITERATOR in der Automatisierungsdefinition für den Zugriff freigegeben werden.

Makro	Bedeutung
AUTOITERATOR(state, init, test, step, extract)	Implementiert einen Kollektionsiterator innerhalb der automatisierten Klasse.
AUTOITERATOR_DECLARE(state)	Derklariert einen Kollektionsiterator innerhalb der automatisierten Klasse, implementiert diesen jedoch nicht. Verwenden Sie dieses Makro, wenn die Implementierung Ihres Iterators zu komplex für AUTOITERATOR ist.

Die fünf Argumente von AUTOITERATOR definieren den Iterierungsalgorithmus für die Kollektionsklasse. In einer Klasse kann es immer nur einen Autoiterator geben, deshalb muß heir kein eigener interner Name vergeben werden. Die fünf Argumente stellen je einen Code-Abschnitt dar, und ihre Reihenfolge entspricht der Codesequenz in einer **for**-Schleife. Wie das Beispiel zeigt, kann das Iteratorobjekt Klassenelemente referenzieren, da es in die automatisierte Kollektionsklasse eingebettet ist.

Parameter	Beispiel	Bedeutung
state	intIndex	Deklaration der Statusvariablen. Diese Deklaration muß mit der zuvor in AUTOITERATOR_DECLARE gegebenen übereinstimmen.
init	Index = 0	Anweisungen (normalerweise Zuweisungen), die zur Initialisierung der Schleife ausgeführt werden.
test	Index < This-> Total	Boolescher Ausdruck, der bei jedem Durchgang durch die Schleife getestet wird.
step	Index++	Anweisungen, die bei jedem Durchgang durch die Schleife ausgeführt werden.
extract	(This-> Array)[Index]	Ausdruck, der die nachfolgenden Objekte in der Kollektion zurückliefert.

Innerhalb der Parameter zeigt *This* (beachten Sie das große T) auf das einschließende Kollektionsobjekt, nicht das eingebettete Iteratorobjekt.

Kommata können nur innerhalb runder Klammern verwendet werden. Semikolon kann zur Trennung mehrfacher Anweisungen verwendet werden, aber nicht zum Beenden eines Makro-Arguments.

Wenn Sie AUTOITERATOR_DECLARE anstelle von AUTOITERATOR verwenden, müssen Sie die folgenden Statusvariablen und Methoden implementieren, entsprechend den für AUTOITERATOR beschriebenen Schritten.

```
void Init();
bool Test();
void Step();
void Return(TAutoVal& v);
```

Makros AUTONAMES ocf/automacr.h

Die AUTONAMES-Makros stellen den ersten von drei Makrosätzen dar, die ein Automatisierungskontroller zur Implementierung von Methoden in seinen Platzhalterobjekten verwendet. AUTONAMES-Makros weisen beliebigen Argumenten, die der Kontroller durch Namen referenzieren möchte, Namen zu. Benannte Parameter haben Standardwerte und müssen in Befehlen nicht übergeben werden. Wenn ein Befehl fünfzehn Parameter hat, von denen zehn benannt sind und über Standardwerte verfügen, dann muß der Kontroller die fünf nicht benannten Parameter in jedem Fall übergeben, kann aber bei den verbleibenden zehn die Auswahl auf eine beliebige Teilmenge beschränken, wobei er die Identifizierung über den Namen vornimmt.

Der zweite Makrosatz, AUTOARGS, beschreibt die Datentypen nicht benannter Argumente, die im Befehl immer übergeben werden müssen. Der

dritte Satz, AUTOCALL, informiert darüber, ob der Befehl eine Methode oder eine Eigenschaft ist, und was er zurückliefert.

In den folgenden Makros ist *id* die numerische Kennung einer Methode, *fname* eine Zeichenkette, die eine Methode benennt und *n1, n2, n3* und *n4* sind Zeichenketten, die als Argumentnamen zugewiesen werden. Die meisten der Makros benötigen den Funktionsnamen zur Identifikation der Funktion, wenn aber eine Funktion keine benannten Argumente besitzt, kann statt ihres Namens auch ihre Identifikationsnummer übergeben werden.

Der Header automacr.h definiert AUTONAMES-Makros, die bis zu vier Argumente annehmen. Zur Generierung von Versionen, die mehr als vier Argumente annehmen, wird das Dienstprogramm MACROGEN.EXE verwendet.

Makro	Bedeutung
AUTONAMES0(id)	Die Funktion *id* hat keine benannten Argumente.
AUTONAMES0(fname)	Die Funktion *fname* hat keine benannten Argumente.
AUTONAMES1(fname, n1)	Die Funktion *fname* hat ein benanntes Argument, *n1*.
AUTONAMES2(fname, n1, n2)	Die Funktion *fname* hat zwei benannte Argumente, *n1* und *n2*.
AUTONAMES3(fname, n1, n2, n3)	Die Funktion *fname* hat drei benannte Argumente, *n1*, *n2* und *n3*.
AUTONAMES4(fname, n1, n2, n3, n4)	Die Funktion *fname* hat vier benannte Argumente, *n1*, *n2*, *n3* und *n4*.

Makro AUTONOHOOK ocf/automacr.h

AUTONOHOOK

Ein Automatisierungs-Server verwendet AUTONOHOOK in einem Automatisierungsdeklarations-Makro, um jegliches Einschalten von Hook-Prozeduren in den Befehl zu verhindern. Nicht einmal ObjectComponents kann den Aufruf überwachen.

AUTONOHOOK ist nur in sehr hochentwickelten Anwendungen einzusetzen.

Makros AUTOPROP ocf/automacr.h

Ein Automatisierungs-Server verwendet AUTOPROP-Makros in seiner Automatisierungsdeklaration (nach DECLARE_AUTOCLASS), um Eigenschaften einer automatisierten Klasse für den Zugriff durch OLE zugänglich zu machen. Eine Eigenschaft ist ein Datum, das nicht direkt gelesen oder geschrieben werden kann, sondern nur durch eine Menge von Zugriffsfunktionen (wie beispielsweise *GetPosition* und *SetPosition*).

Ein Server kann diese Zugriffsfunktionen in jeder von ihm gewünschten Weise implementieren. Da nur die Zugriffsfunktionen zugänglich sind, muß die Eigenschaft kein Datenelement sein. Um im Beispiel zu bleiben, *GetPosition* und *SetPosition* müßten sich nicht auf ein Datenelement vom Typ *TPoint* beziehen. Sie könnten die Cursorposition stattdessen beim System abfragen und die Antwort zurückliefern.

Die drei AUTOPROP-Makros haben ähnliche Parameter. *name* ist ein interner Name, den Sie der Eigenschaft zuweisen. ObjectComponents verwendet den Namen, um die automatisierten Elemente zu verwalten. Dieser interne Name wird sonst nur noch in der entsprechenden Automatisierungsdefinition verwendet.

get und *set* sind die Zugriffsfunktionen. Eine Eigenschaft, die nur Lesezugriff gestattet, hat nur die get-Funktion. Eine Eigenschaft, die nur Schreibzugriff gestattet, hat nur die set-Funktion.

type ist der Datentyp der Eigenschaft. Normalerweise ist das ein C++ Datentyp, aber Strings und Objekteigenschaften sind in besonderer Weise zu behandeln. Näheres finden Sie unter Automatisierungsdatentypen.

In *options* kann eine Hook-Prozedur eingeschaltet werden. Der Code einer Hook-Prozedur wird bei jeder Ausführung des Automatisierungsbefehls aufgerufen. Hook-Prozeduren können Befehle aufzeichnen, sie rückgängig machen oder validieren. Näheres finden Sie unter Hook-Makros für die Automatisierung. Der Parameter *options* kann weggelassen werden, doch muß dem vorangehenden Argument in jedem Fall ein Komma folgen.

Makro	Bedeutung
AUTOPROP(name, get, set, type, options)	Die Eigenschaft kann gelesen und geschrieben werden.
AUTOPROP(name, get, type, options)	Die Eigenschaft kann nur gelesen, aber nicht berändert werden.
AUTOPROP(name, set, type, options)	Die Eigenschaft kann geändert, aber nicht gelesen werden (selten).

Makro AUTORECORD ocf/automacr.h

AUTORECORD(code)

Ein Automatisierungs-Server verwendet AUTORECORD in einem Automatisierungsdeklarations-Makro, um benutzerdefinierten Code einzuschalten (Hook-Prozedur), die alle Aufrufe verzeichnet, die an einem bestimmten Automatisierungsbefehl ergehen. *code* ist der Ausdruck oder der Funktionsaufruf, der bei jedem Befehl ausgeführt werden soll. Er sollte alle Informationen speichern, die die Anwendung benötigt, um den gleichen Befehl zu einem späteren Zeitpunkt wiederzugeben.

Das Aufzeichnen wird in der aktuellen Version von ObjectComponents nicht unterstützt.

Makro AUTOREPORT ocf/automacr.h

AUTOREPORT(code)

Ein Automatisierungs-Server verwendet AUTOREPORT in einem Automatisierungsdeklarations-Makro, um benutzerdefinierten Code einzuschalten (Hook-Prozedur), der den Fehlercode einer automatisierten Elementfunktion überprüft. Wenn sich *code* zu 0 ergibt, geht OLE von einer erfolgreichen Ausführung des Befehls aus. Nimmt code einen Wert ungleich 0 an, löst OLE im Kontroller eine Exception aus. Innerhalb des Code-Ausdrucks wird *Val* zum Bezug auf den tatsächlichen Rückgabewert verwendet.

Aufzählungstyp AutoSymFlag ocf/autodefs.h

enum AutoSymFlag

Diese Flags werden in der Klasse *TAutoCommand* verwendet, um Attribute eines Automatisierungsbefehls zu beschreiben. Diese Flags informieren darüber, ob ein Befehl eine Methode oder eine Eigenschaft ist, ob Argumente über den Wert oder durch Referenz übergeben werden, und ob der Befehl in Symbolanzeigen zur Typinformation sichtbar sein soll.

Konstante	Bedeutung
asAnyCommand	Beliebiger Befehl: Methode, Zugriff auf Eigenschaften, Objektgenerator.
asOleType	Methode oder Eigenschaft, die für OLE zugänglich ist.
asMethod	Methode.
asGet	Liefert den Wert einer Eigenschaft zurück.
asIterator	Iterator-Eigenschaft; wird verwendet, um Elemente in einer Kollektion aufzuzählen.
asSet	Wert einer Eigenschaft setzen.
asGetSet	Wert einer Eigenschaft abfragen oder setzen.
asBuild	Konstruktorbefehl (in OLE 2.01 nicht unterstützt).
asArgument	Eigenschaft, die ein Objekt zurückgibt.
asArgByVal	Der Wert des Arguments wird übergeben.
asArgByRef	Das Argument ist ein Zeiger auf einen Wert.
asFactory	Zum Erzeugen von Objekten oder Determinieren von Klassen.
asClass	Erweiterung auf eine weitere Klassensymbol-Tabelle.
asBindable	Sendet die Benachrichtigung *OnChanged*.
asRequestEdit	Sendet *OnRequest* editieren vor der Änderung
asDisplayBind	Diese Eigenschaft ist als einzige voreingestellt (redundant).
asHidden	In der normalen Symbolanzeige nicht sichtbar.
asPersistent	Eigenschaft ist persistent.

Siehe auch: Öffentlicher Konstruktor TAutoCommand

Makros AUTOSTAT ocf/automacr.h

Verwenden Sie ein AUTOSTAT-Makro in einer Automatisierungsdeklaration (nach DECLARE_AUTOCLASS), um statische Elementfunktionen und globale Funktionen für den Zugriff durch OLE zugänglich zu machen.

Alle Versionen des Makros AUTOSTAT haben ähnliche Parameter. *name* ist ein interner Name, den Sie der Funktion zuweisen. ObjectComponents verwendet den Namen zur Verwaltung der automatisierten Elemente. Dieser interne Name wird sonst nur noch in der entsprechenden Automatisierungsdefinition verwendet.

func ist der Name der statischen oder globalen Funktion.

ret ist der Typ des Werts, den die Funktion zurückgibt.

type1, t1, t2, t3 und *t4* sind die Typen der Argumente der Funktion.

In *options* kann eine Hook-Prozedur eingeschaltet werden. Der Code einer Hook-Prozedur wird bei jeder Ausführung des Automatisierungsbefehls aufgerufen. Hook-Prozeduren können Befehle aufzeichnen, sie rückgängig machen oder validieren. Näheres finden Sie unter Hook-Makros für die Automatisierung. Der Parameter *options* kann weggelassen werden, doch muß dem vorangehenden Argument in jedem Fall ein Komma folgen.

Die Rückgabetypen und Argumenttypen sind für gewöhnlich normale C++ Datentypen, aber Strings und Objektwerte erfordern eine besondere Behandlung. Näheres hierzu finden Sie unter Automatisierungsdatentypen.

Makro	Bedeutung
AUTOSTAT0(name, func, ret, options)	Der statischen Funktion *func* wird das Symbol *name* zugewiesen. Sie nimmt keine Argumente an und liefert einen Wert vom Typ *ret* zurück.
AUTOSTAT0V(name, func, options)	Die statische Funktion *func* nimmt keine Argumente an und gibt keinen Wert zurück.
AUTOSTAT1(name, func, ret, type1, options)	*func* akzeptiert ein Argument vom Typ *type1* und liefert einen Wert vom Typ *ret* zurück.
AUTOSTAT1V(name, func, type1, options)	*func* akzeptiert ein Argument vom Typ *type1* und liefert keinen Wert zurück.
AUTOSTAT2(name, func, ret, t1, t2, options)	*func* akzeptiert zwei Argumente der Typen *t1* und *t2* und liefert einen Wert vom Typ *ret* zurück.
AUTOSTAT2V(name, func, t1, t2, options)	*func* akzeptiert zwei Argumente und gibt keinen Wert zurück.

AUTOSTAT3(name, func, ret, t1, t2, t3, options)	*func* akzeptiert drei Argumente und gibt einen Wert zurück.
AUTOSTAT3V(name, func, t1, t2, t3, options)	*func* akzeptiert drei Argumente und gibt keinen Wert zurück.
AUTOSTAT4(name, func, ret, t1, t2, t3, t4, options)	*func* akzeptiert vier Argumente und gibt einen Wert zurück.
AUTOSTAT4(name, func, t1, t2, t3, t4, options)	*func* akzeptiert vier Argumente und gibt keinen Wert zurück.

Makro AUTOTHIS ocf/automacr.h

AUTOTHIS(name, type, options)

Ein Automatisierungs-Server verwendet das Makro AUTOTHIS in seiner Automatisierungsdeklaration (nach DECLARE_AUTOCLASS), wenn er das C++ Objekt selbst als ein Mitglied des automatisierten OLE-Objekts zugänglich machen will.

name ist ein interner Name, den Sie der Eigenschaft zuweisen. ObjectComponents verwendet den Namen zur Verwaltung der automatisierten Elemente. Dieser interne Name findet sich darüber hinaus nur noch in der korrespondierenden Automatisierungsdefinition.

type muß *TAutoObject<T>* sein, wobei *T* der Typ der automatisierten Klasse ist.

In *options* kann eine Hook-Prozedur eingeschaltet werden. Der Code der Hook-Prozedur wird dann jedesmal ausgeführt, wenn der Kontroller wegen dieser Eigenschaft anfragt. Hook-Prozeduren können Befehle aufzeichnen, sie rückgängig machen oder validieren. Näheres finden Sie unter Hook-Makros für die Automatisierung. Der Parameter *options* kann weggelassen werden, doch muß dem vorangehenden Argument in jedem Fall ein Komma folgen.

Makro AUTOUNDO ocf/automacr.h

AUTOUNDO(code)

Ein Automatisierungs-Server verwendet AUTOUNDO in einem Automatisierungsdeklarations-Makro, um benutzerdefinierten Code einzuschalten (Hook-Prozedur), der alle Informationen aufzeichnet, die die Anwendung benötigt, um den Befehl später rückgängig zu machen. Normalerweise schiebt dieses Makro Informationen auf einen vom Benutzer verwalteten Undo-Stack. Zu diesen Informationen können beispielsweise auch die Parameter gehören, die die Umkehrung des ursprünglichen Befehls ausführen. Um eine Serie von Aktionen rückgängig zu machen, kann das Programm Befehle vom Undo-Stack holen und sie ausführen. *code* ist der Ausdruck oder die Funktion, der/die die Informationen aufzeichnet.

Das Rückgängigmachen von Befehlen wird von der aktuellen Version von ObjectComponents nicht unterstützt.

Makro AUTOVALIDATE ocf/automacr.h

AUTOVALIDATE(condition)
Ein Automatisierungs-Server verwendet AUTOVALIDATE in einem Automatisierungsdeklarations-Makro, um benutzerdefinierten Code einzuschalten (Hook-Prozedur), der die Gültigkeit von empfangenen Argumenten bestätigt, bevor sie zur Verarbeitung in einem Befehl weitergegeben werden. *condition* ist ein Ausdruck oder eine Funktion, der/die sich zu **true** ergibt, wenn die empfangenen Argumente für den Befehl gültig sind und zu **false**, wenn das nicht der Fall ist. Wenn der Ausdruck **false** zurückliefert, löst OLE eine Exception in der Kontrolleranwendung aus.

Registrierschlüssel clsid

Registriert einen global eindeutigen Bezeichner (globally unique identifier, GUID) für die Klassen-ID der Anwendung. Ein GUID ist ein 16-Bit-Wert, der als String dargestellt werden kann.

Ein *clsid* GUID ist für jede Anwendungs-Registriertabelle erforderlich. Weitere festzulegen ist niemals erforderlich. Wenn weitere GUIDs für Ihre Dokumente, die Typbibliothek, automatisierte Klassen oder Debuggeraufrufe benötigt werden, stellt ObjectComponents diese automatisch zur Verfügung, indem es das niederwertige Feld des ersten GUID inkrementiert. Stellen Sie sicher, daß Sie den gesamten Bereich von Zahlen, den Ihre Anwendung tatsächlich benötigt, berücksichtigen, wenn Sie den nächsten verfügbaren GUID für ein anderes Programm festlegen.

Ein *clsid* kann auf verschiedene Weise zur Verfügung gestellt werden. Zum einen kann das Dienstprogramm GUIDGEN im Verzeichnis OCTOOLS ausgeführt werden. Auch AppExpert generiert automatisch einen GUID für alle von ihm erzeugten OLE-fähigen Anwendungen. Eine andere Möglichkeit, einen GUID zu erhalten, besteht im Aufrufen von *GoCreateGuid* des OLE-API, wie in der Hilfedatei OLE2HELP.HLP beschrieben. Schließlich können Sie sich an Microsoft wenden und sich einen Block von GUIDs permanent zuweisen lassen.

Jede Anwendung muß über einen eigenen, absolut eindeutigen *clsid*-String verfügen, also verwenden Sie niemals Werte, die Sie aus Beispielprogrammen kopiert haben.

Zum Registrieren eines *clsid* wird das Makro REGDATA verwendet. Dabei ist *clsid* der erste Parameter, der zweite ist ein GUID-String.

```
REGDATA(clsid, "{CDE7F941-544B-101B-A9C1-04021C007002}")
```

Registrierschlüssel cmdline

Registriert Argumente, die OLE beim Starten der Programmdatei des Servers in die Kommandozeile schreiben sollte.

Der Schlüssel *cmdline* ist in Anwendungs-Registriertabellen gültig, wird aber bei DLL-Servern ignoriert. Jede Anwendung kann ihn registrieren, aber normalerweise haben nur Automatisierungs-Server einen Nutzen davon.

Automatisierungs-Server können den Schlüssel *cmdline* verwenden, um den Schalter *-Automation* zu setzen. Wenn das Registrar-Objekt auf diesen Schalter trifft, setzt es die registrierte *usage*-Einstellung der Anwendung außer Kraft, und erzwingt die Ausführung des Programms im Einbenutzer-Modus. Bei Servern, die sowohl Verknüpfung und Einbettung (Linking and Embedding) als auch Automatisierung unterstützen, ist dieses Vorgehen sinnvoll. Als Verknüpfungs- und Einbettungs-Server kann er eventuell mit einer einzigen Instanz parallel ausgeführte Client-Anwendungen unterstützen. Arbeitet der Server aber als Automatisierungs-Server, können die meisten Anwendungen es nicht akzeptieren, wenn parallel ausgeführte Client-Programme die gleiche Instanz eines Objektes steuern.

Zum Registrieren von Kommandozeilenoptionen wird das Makro REGDATA mit *cmdline* als erstem Parameter und einer Zeichenkette, die die Kommandozeilenargumente enthält, als zweitem Parameter, verwendet.

```
REGDATA(cmdline, "/automation")
```

Registrierschlüssel debugclsid

Ein GUID, der die Debug-Version eines Servers identifiziert. Sie sollten diesen Schlüssel niemals direkt registrieren. Er wird immer automatisch beim Registrieren von *debugprogid* generiert.

Dieser Schlüssel wird von DLL-Servern ignoriert.

Siehe auch: Registrierschlüssel debugprogid

Registrierschlüssel debugdesc

Ein String, der die Debug-Version Ihres Programms beschreibt. Wenn er bei der Registrierung eines Dokuments verwendet wird, erscheint dieser String im Menü Objekt Einfügen. Wenn er beim Registrieren von Anwendungen verwendet wird, erscheint er in Objekt-Symbolanzeigen. Der String kann bis zu vierzig Zeichen enthalten und ist lokalisierbar.

Der Schlüssel *debugdesc* wird in den Anwendungs- und Dokument-Registriertabellen aller Programme benötigt, die den Schlüssel *debugprogid* registrieren. Darüber hinaus hat er keine Bedeutung.

Zum Registrieren des Schlüssels *debugdesc* wird das Makro REGDATA verwendet. *debugdesc* wird als erster Parameter übergeben und der beschreibende String als zweiter.

```
REGDATA(debugdesc, "Meine Anwendung (debugging)")
```

Dieser Schlüssel wird von DLL-Servern ignoriert.

Siehe auch: Registrierschlüssel debugclsid, Registrierschlüssel debugger, Registrierschlüssel debugprogid, Makro REGDATA

Registrierschlüssel debugger

Registriert den Pfad und den Dateinamen für das Laden Ihres Debuggers.

Der Schlüssel *debugger* ist in jeder Registriertabelle gültig. Er muß immer dann angegeben werden, wenn auch *debugprogid* registriert wird. Darüber hinaus hat er keine Bedeutung.

Zum Registrieren des Schlüssels *debugger* wird das Makro REGDATA verwendet. *debugger* wird als erster Parameter übergeben, den zweiten bildet der Kommandozeilen-String für die Debugger-Anwendung. Wenn OLE den Debugger aufruft, plaziert er den zweiten Parameter-String in der Kommandozeile vor dem .EXE-Pfad des Servers. Der Debugger-String kann optional einen vollständigen Pfad und Kommandozeilenschalter des Debuggers enthalten.

```
REGDATA(debugger, "TDW") // vorausgesetzt, TDW befindet sich im aktuellen
Pfad
```

Dieser Schlüssel wird von DLL-Servern ignoriert.

Siehe auch: Registrierschlüssel debugclsid, Registrierschlüssel debugdesc, Registrierschlüssel debugprogid, Makro REGDATA

Registrierschlüssel debugprogid

Ein String, der die Debug-Version eines Programms identifiziert. Ebenso wie ein *progid*-String besteht auch dieser String aus zwei Teilen, die durch einen Punkt getrennt sind. Der erste Teil ist der Name Ihres Programms und der zweite ist *.debug*.

Die Zuweisung eines Wertes zum Schlüssel *debugprogid* veranlaßt ObjectComponents, für den Server zwei Sätze von Einträgen in der Registrierdatenbank zu erzeugen. Wenn Sie im Menü Bearbeiten den Eintrag Objekt Einfügen anwählen, erscheinen beide Einträge in der Liste. Wenn Sie den Debugger-Eintrag auswählen, ruft ObjectComponents Ihren Debugger zusammen mit dem Server auf. Ohne die Möglichkeit, einen doppelten Debugger-Eintrag zu registrieren, wäre es schwierig, den Server zu debuggen, wenn OLE ihn aufruft.

ObjectComponents erzeugt automatisch eine *clsid* für den Debuggereintrag.

Der Schlüssel *debugprogid* ist in Anwendungs-Registriertabellen optional und für Dokument-Registriertabellen ohne Bedeutung. Wenn Sie *debugprogid* registrieren, müssen Sie auch *debugdesc* und *debugger* registrieren.

Zum Registrieren eines *debugprogid*-Schlüssels wird das Makro REGDATA verwendet. Dabei ist *debugprogid* der erste Parameter und der ID-String der zweite.

```
REGDATA(debugprogid, "MeinProg.Debug")
```

Dieser Schlüssel wird von DLL-Servern ignoriert.

Siehe auch: Registrierschlüssel debugclsid, Registrierschlüssel debugdesc, Registrierschlüssel debugger, Registrierschlüssel progid, Makro REGDATA

Makro DECLARE_AUTOCLASS ocf/automacr.h

DECLARE_AUTOCLASS(cls)

DECLARE_AUTOCLASS(cls) leitet einen Block von Makros ein, die automatisierbare Elemente der benutzerdefinierten Klasse *cls* deklarieren.

Ein Automatisierungs-Server verwendet DECLARE_AUTOCLASS, um einen Block von Makros einzuleiten, die automatisierbare Elemente einer benutzerdefinierten C++ Klasse für den Zugriff durch OLE zugänglich machen. *cls* ist der Name dieser C++ Klasse.

Der Block von Deklarationsmakros erscheint für gewöhnlich in der Definition der automatisierbaren C++ Klasse. Ein entsprechender Block von

Automatisierungsdefinitions-Makros muß in der Implementation der automatisierbaren C++ Klasse vorkommen.

Makros DECLARE_COMBASES*n* ocf/oleutil.h

Verwenden Sie die COMBASES-Makros zur Erzeugung von C++ Objekten, die dem OLE Component Object Model (COM) entsprechen. COM-Objekte unterstützen OLE-Schnittstellen und gestatten Ihnen das Ableiten von Klassen, die direkt mit OLE zusammenarbeiten, statt über ObjectComponents. Diese Makros sind für erfahrene Anwender bestimmt.

COM-Objekte werden als Basisklassen für andere Objekte benutzt. Die abgeleitete Klasse muß sowohl von Ihrer COM-Klasse als auch von der Klasse *TUnknown* von ObjectComponents erben. *TUnknown* inplementiert die Steuerungsschnittstelle *IUnknown* für Ihr Objekt.

Zum Erzeugen einer COM-Klasse:

Stellen Sie der Deklaration der Klasse eines der DECLARE_COMBASES-Makros voran. Welches Makro Sie wählen hängt davon ab, wie viele Schnittstellen (neben *IUnknown*) die COM-Klasse unterstützt.

Stellen Sie der Implementation Ihrer COM-Klasse das entsprechende DEFINE_COMBASES-Makro voran.

Leiten Sie Ihre endgültige Klasse mehrfach aus *TUnknown* und aus Ihrer neuen COM-Klasse ab (in dieser Reihenfolge).

Das erste Makro-Argument ist *name*. Das ist der Name der COM-Klasse, die Sie erzeugen. *i1, i2, i3* und *i4* sind die Namen der Schnittstellen, die Ihre COM-Klasse unterstützt.

Makro	Bedeutung
DECLARE_COMBASES1(name, i1)	Deklariert eine COM-Klasse *name*, die von der Schnittstellenklasse *i1* erbt.
DECLARE_COMBASES2(name, i1, i2)	Deklariert eine COM-Klasse *name*, die von den Schnittstellenklassen *i1* und *i2* erbt.
DECLARE_COMBASES3(name, i1, i2, i3)	Deklariert eine COM-Klasse *name*, die von den Schnittstellenklassen *i1, i2* und *i3* erbt.
DECLARE_COMBASES4(name, i1, i2, i3, i4)	Deklariert eine COM-Klasse *name*, die von den Schnittstellenklassen *i1, i2, i3* und *i4* erbt.

Makro DEFINE_AUTOAGGREGATE ocf/automacr.h

DEFINE_AUTOAGGREGATE(cls, AggregatorFunction)
Ein Automatisierungs-Server verwendet DEFINE_AUTOAGGREGATE, um einen Block von Makros einzuleiten, die automatisierbare Elemente von *cls* definieren, einer benutzerdefinierten C++ Klasse. Der Block endet mit dem Makro END_AUTOAGGREGATE.

DEFINE_AUTOCLASS arbeitet prinzipiell genauso, verfügt aber nicht über den zusätzlichen Parameter *AggregatorFunction*. Verwenden Sie DEFINE_AUTOAGGREGATE, wenn die C++ Klasse, die Sie automatisieren, ein Component Object Model (COM)-Objekt ist, von einem solchen erbt oder an ein solches delegiert. Der Parameter *AggregatorFunction* zeigt auf die Aggregatfunktion zum Erreichen des COM-Objekts. Wenn der Name des COM-Objekts beispielsweise *Aggregate* ist, könnte die Aggregatfunktion jeder der folgenden Ausdrücke sein:

```
Aggregate       //das automatisierte C++ Objekt ist das COM-Objekt
OcApp->Aggregate //das automatisierte C++ Objekt delegiert an das
COM-Objekt
MyBase::Aggregate  //das automatisierte C++ Objekt erbt von dem COM-Objekt
```

Siehe auch: DEFINE_AUTOCLASS, END_AUTOAGGREGATE

Makro DEFINE_AUTOCLASS ocf/automacr.h

DEFINE_AUTOCLASS(cls)
Leitet einen Block von Makros in einem Automatisierungs-Server ein. Jedes Makro in dem Block definiert ein automatisierbares Element von *cls*, einer benutzerdefinierten C++ Klasse. Der Block endet mit dem Makro END_AUTOCLASS.

Der Block von Definitionsmakros erscheint in der Implementierung der automatisierbaren C++ Klasse. Ein entsprechender Block von Automatisierungsdeklarations-Makros muß in der Definition derselben Klasse vorkommen.

Siehe auch: Makro END_AUTOCLASS

Makros DEFINE_COMBASES*n* ocf/oleutil.h

Implementiert die Schnittstelle *IUnknown* für jede der OLE-Schnittstellen, die Ihr COM-Objekt unterstützt.

Verwenden Sie die COMBASE-Makros zum Erzeugen von C++ Objekten, die dem OLE Component Object Model (COM) entsprechen. COM-Objekte unterstützen OLE-Schnittstellen und erlauben die Ableitung von Klassen, die direkt mit OLE zusammenarbeiten, statt über ObjectComponents. Diese Makros sind für erfahrene Anwender bestimmt.

Automatisierte Objekte können über die Makros DEFINE_AUTOAGGREGATE und DECLARE_AUTOAGGREGATE an COM-Objekte delegieren.

Zum Erzeugen einer COM-Klasse:

Stellen Sie der Deklaration der Klasse eines der DECLARE_COMBASES-Makros voran. Welches Makro Sie wählen hängt davon ab, wie viele Schnittstellen (neben *IUnknown*) die COM-Klasse unterstützt.

Stellen Sie der Implementation Ihrer COM-Klasse das entsprechende DEFINE_COMBASES-Makro voran.

Leiten Sie Ihre endgültige Klasse mehrfach aus *TUnknown* und aus Ihrer neuen COM-Klasse ab (Mehrfachvererbung).

COM-Objekte können an andere COM-Objekte delegieren und verwenden dazu einen anderen zusammenhängenden Makrosatz, der ebenfalls in oleutil.h definiert ist. Nähere Informationen finden Sie in der Header-Datei für die Makros DEFINE-QI-xxxx.

Makro	Bedeutung
DEFINE_COMBASES1(name, i1)	*IUnknown* für die Schnittstelle *i1* in der COM-Klasse *name* definieren.
DEFINE_COMBASES2(name, i1, i2)	*IUnknown* für die Schnittstellen *i1* und *i2* in der COM-Klasse *name* definieren.
DEFINE_COMBASES3(name, i1, i2, i3)	*IUnknown* für die Schnittstellen *i1*, *i2* und *i3* in der COM-Klasse *name* definieren.
DEFINE_COMBASES4(name, i1, i2, i3, i4)	*IUnknown* für die Schnittstellen *i1*, *i2*, *i3* und *i4* in der COM-Klasse *name* definieren.

Siehe auch: Makros DECLARE_COMBASES*n*

Registrierschlüssel description

Registriert einen beschreibenden Namen von bis zu 40 Zeichen Länge, der dem Anwender angezeigt werden soll. Diese Zeichenkette beschreibt die Anwendung oder ihre Dokumenttypen und erscheint im Dialogfenster Objekt Einfügen. Dieser Wert sollte lokalisiert werden.

In jeder Registriertabelle wird eine solche Beschreibungs-Zeichenkette benötigt. Ein Beschreibungs-String wird mit dem Makro REGDATA registriert. Dabei wird *description* als erster Parameter übergeben und der Beschreibungs-String als zweiter.

```
REGDATA(description, "OWL Drawing Pad 2.5")
```

Siehe auch: Makro REGDATA

Registrierschlüssel directory

Registriert das Standardverzeichnis für Dokumentdateien.Das Dokument-Klassen-Template bezieht sich auf diesen Pfad, wenn es das allgemeine Dialogfenster Datei Öffnen aufruft. Der Verzeichnispfad wird von OLE nicht verwendet.

Der Schlüssel directory ist in allen Dokument-Registriertabellen gültig. Er ist immer optional.

Zum Registrieren eines Verzeichnisses wird das Makro REGDATA verwendet. directory wird als erster Parameter übergeben und der Pfadname als zweiter.

```
REGDATA(directory, "C:\\temp")
```

Registrierschlüssel docfilter

Registriert eine Dateispezifikation für das Auflisten von Dateien, die von der Anwendung erzeugt werden. Diese Information wird vom Dokument-Klassen-Template beim Erzeugen des allgemeinen Dialogfensters Datei Öffnen verwendet. OLE wertet die Information nicht aus.

docfilter ist in allen Dokument-Registriertabellen gültig. Es muß immer angegeben werden, es sei denn, im entsprechenden *docflag*-Schlüssel ist das Flag *dtHidden* gesetzt. Wenn Sie einen Dokumentfilter registrieren, möchten Sie wahrscheinlich auch den Schlüssel *extension* registrieren.

Zum Registrieren eines Dokumentfilters wird das Makro REGDATA verwendet, wobei *docfilter* den ersten und ein Filterstring den zweiten Parameter bildet.

```
REGDATA(docfilter, "*.txt")
```

Siehe auch: Registrierschlüssel docflags, Dokument-View-Konstanten dt, Registrierschlüssel extension, Makro REGDATA

Registrierschlüssel docflags

Registriert Options-Flags für die Dokument-View der Dokumente einer Anwendung. Diese Information wird nur vom Dokument-Klassen-Template ausgewertet, nicht von OLE. Das Dokument-Template verwendet die Flags, um seine Anzeige des allgemeinen Dialogfensters Datei Öffnen

zu steuern. Eine Liste aller Flags befindet sich in der Beschreibung der Dokument-View-Konstanten *dtxxxx*.

Der Schlüssel *docflags* ist in allen Dokument-Registriertabellen gültig und ist immer optional.

Zum Registrieren von Dokument-Flags wird das Makro REGDOCFLAGS verwendet.

```
REGDOCFLAGS(dtAutoOpen | dtAutoDelete | dtUpdateDir | dtCreatePrompt)
```

Siehe auch: Registrierschlüssel docfilter, Dokument-View-Konstanten dt, Makro REGDOCFLAGS

Funktion DynamicCast ocf/autodefs.h

const void far* DynamicCast(const void far* obj, const typeinfo& scr, const typeinfo& dst);
Versucht, einen Zeiger aus einem Typ in einen anderen umzuwandeln. Der Versuch gelingt nur, wenn der alte Typ und der neue Typ durch Vererbung mit einander verwandt sind.

obj ist der Zeiger, der in einen neuen Typ umgewandelt werden soll. *scr* ist Typinformation über das Quellobjekt. *dst* ist Information über den Zieltyp.

Wenn die Umwandlung gelingt, liefert *DynamicCast* einen Zeiger auf den neuen Typ zurück. Schlägt sie fehl, ist der Rückgabewert Null.

Die Parameter *scr* und *dst* können mit dem Parameter *typeid* generiert werden. (Für ObjectComponents muß RTTI verwendet werden).

Siehe auch: Funktion MostDerived, typeid, Klasse typeinfo

Makro END_AUTOAGGREGATE ocf/automacr.h

END_AUTOAGGREGATE(cls, name, doc, help)
Beendet einen Block von Makros, die ein Automatisierungs-Server zur Definition automatisierbarer Methoden in *cls* verwendet, einer benutzerdefinierten C++ Klasse. Der Definitionsblock beginnt mit DEFINE_AUTOAGGREGATE.

Die verwandten Makros DEFINE_AUTOCLASS und END_AUTOCLASS markieren ebenfalls einen Block von Automatisierungsdefinitionen. Verwenden Sie die Aggregate-Makros, wenn *cls* ein Component Object Model (COM)-Objekt ist, von einem solchen erbt oder an es delegiert.

name ist die Zeichenkette, die Automatisierungskontroller verwenden, um Objekte vom Typ *cls* zu identifizieren. Wenn der Anwender bei OLE den Objektnamen abfragt, gibt das System die Zeichenkette in *doc* zurück. Wenn für das Objekt eine .HLP-Datei registriert ist, dann zeigt die Kontext-ID in *help* auf einen Hilfebildschirm, der das Objekt beschreibt.

Siehe auch: Makro DEFINE_AUTOAGGREGATE, Makro END_AUTOCLASS

Makro END_AUTOCLASS ocf/automacr.h

END_AUTOCLASS(cls, name, doc, help)
Beendet den Block von Makros, die ein Automatisierungs-Server für die Definition automatisierbarer Methoden in *cls* verwendet, einer benutzerdefinierten C++ Klasse. Der Definitionsblock beginnt mit DEFINE_AUTOCLASS.

name ist der String, den Automatisierungskontroller zur Identifikation von Objekten vom Typ *cls* verwenden. Wenn der Benutzer den Objektnamen bei OLE abfragt, liefert das System den String in *doc* zurück. Wenn für das Objekt eine .HLP-Datei registriert ist, dann zeigt die Kontext-ID in *help* auf einen Hilfebildschirm, der das Objekt beschreibt.

Siehe auch: Makro DEFINE_AUTOCLASS

Makro EXPOSE_APPLICATION ocf/automacr.h

EXPOSE_APPLICATION(cls, extName, doc, help)
Ein Automatisierungs-Server verwendet EXPOSE_APPLICATION in seiner Automatisierungsdefinition (nach DEFINE_AUTOCLASS), wenn er die Anwendung selbst als Mitglied ihres automatisierten Objekts für den Zugriff durch OLE zugänglich machen will. In den OLE-Konventionen wird ausdrücklich nahegelegt, daß jedes Automatisierungsobjekt dieses Element enthalten sollte.

cls ist der Klassenname der Anwendung.

extName ist der externe, öffentliche Name, den Sie diesem Element zuweisen. Automatisierungskontroller verwenden diesen String, um sich auf das Anwendungselement zu beziehen. Der String kann lokalisiert werden.

doc ist ein String, der dem Anwender dieses Element beschreibt. Ein Automatisierungskontroller kann diesen String bei OLE abfragen, wenn der Anwender Hilfe anfordert.

help ist eine Kontext-ID für eine .HLP-Datei. Dieser Parameter ist optional und hat nur dann eine Funktion, wenn auch eine .HLP-Datei zur Dokumentierung des Servers registriert wird. Geschieht dies, dann übergibt der Kontroller diese Kontext-ID an das Hilfesystem, das einen Hilfebildschirm mit der Objektbeschreibung anzeigt, wenn der Benutzer Hilfe anfordert.

Makro EXPOSE_DELEGATE ocf/automacr.h

EXPOSE_DELEGATE(cls, extName, locator)
Ein Automatisierungs-Server verwendet EXPOSE_DELEGATE in seiner Automatisierungsdefinition (nach DEFINE_AUTOCLASS), um zwei nicht aufeinander bezogene C++ Klassen in einem einzigen OLE-Automatisierungsobjekt zusammenzufassen. Im Ergebnis delegiert dann die primäre automatisierte Klasse der Anwendung Aufgaben an eine andere automatisierte Klasse. Dieses Makro weist ObjectComponents an, beide Klassen zu durchsuchen um zu ermitteln, welche Befehle das automatisierte OLE-Objekt ausführen kann.

cls ist der Name der Hilfsklasse, die ebenfalls automatisiert sein muß. Sie muß, mit anderen Worten, ihre eigene Automatisierungsdeklaration und -definition enthalten.

extName ist ein externer, öffentlicher Name, den ein OLE-Kontroller verwendet, um sich auf dieses Element des Objekts zu beziehen.

locator ist eine Funktion, die einen Zeiger auf ein Hilfsobjekt zurückliefert. ObjectComponents benötigt einen Zeiger auf ein Objekt dieses Typs, um Mitglieder dieser Klasse aufzurufen. Die Konvertierung sollte diesem Protoyp entsprechen:

```
auxclass *locator( autoclass *this );
```

Dabei ist *auxclass* der Name der Hilfsklasse und *autoclass* der Name der primären automatisierten Klasse. Im Ergebnis wandelt *locator* einen *this*-Zeiger in einen *that*-Zeiger um.

Siehe auch: Makro EXPOSE_INHERIT

Makro EXPOSE_INHERIT ocf/automacr.h

EXPOSE_INHERIT(cls, extName);
Ein Automatisierungs-Server verwendet EXPOSE_INHERIT in seiner Automatisierungsdefinition (nach DEFINE_AUTOCLASS), um zwei aufeinander bezogene C++ Klassen in einem einzigen OLE-Automat-

isierungsobjekt zusammenzufassen. Im Ergebnis delegiert die primäre automatisierte Klasse der Anwendung Aufgaben an ihre Basisklasse. Dieses Makro weist ObjectComponents an, beide Klassen zu durchsuchen um zu ermitteln, welche Befehle das automatisierte OLE-Objekt ausführen kann.

cls ist der Name der Basisklasse, die ebenfalls automatisiert sein muß. Sie muß, anders ausgedrückt, ihre eigene Automatisierungsdeklaration und -definition enthalten..

ext ist ein externer, öffentlicher Name, den ein OLE-Kontroller zum Bezug auf dieses Element des Objekts verwendet. Er kann lokalisiert werden.

Siehe auch: Makro EXPOSE_DELEGATE

Makro EXPOSE_ITERATOR ocf/automacr.h

EXPOSE_ITERATOR(retType, doc, help);

Ein Automatisierungs-Server verwendet EXPOSE_ITERATOR in seiner Automatisierungsdefinition (nach DEFINE_AUTOCLASS), um ein Iterator-Objekt, das Objekte in einer Kollektion aufzählt, für den Zugriff zugänglich zu machen. Ein Iterator ist nur dann sinnvoll, wenn das automatisierte Objekt selbst eine Kollektion anderer Objekte darstellt. Der Kontroller verwendet Methoden des eingebetteten Iterator-Objekts, um in einer Liste aufeinanderfolgende Objekte zu ermitteln und zu untersuchen.

retType ist ein automatisierter Datentyp, der den Typ der Objekte in der Kollektion beschreibt. Wenn, beispielsweise, die Kollektion ein Array von Shortintegern ist, sollte *retType TAutoShort* sein. Nährere Informationen finden Sie unter Automatisierungsdatentypen.

doc ist eine Zeichenkette, die dem Anwender den Iterator beschreibt. Ein Automatisierungskontroller kann diesen String bei OLE erfragen, wenn der Anwender Hilfe anfordert.

help ist eine Kontext-ID für eine .HLP-Datei. Dieser Parameter ist optional und hat nur dann eine Funktion, wenn auch eine .HLP-Datei zur Dokumentierung des Servers registriert wird. Geschieht dies, dann übergibt der Kontroller diese Kontext-ID an das Hilfesystem, das einen Hilfebildschirm mit der Beschreibung des Iterators anzeigt, wenn der Benutzer Hilfe anfordert.

Siehe auch: Makros AUTOITERATOR

Makros EXPOSE_METHOD ocf/automacr.h

EXPOSE_METHOD(intName, retType, extName, doc, help);
EXPOSE_METHOD_ID(id, intName, retType, extName, doc, help);
Ein Automatisierungs-Server verwendet EXPOSE_METHOD-Makros in seiner Automatisierungsdefinition (nach DEFINE_AUTOCLASS), OLE eine Elementfunktion eines Objekts zur Automatisierung zugänglich zu machen.

intName ist ein interner Name, den Sie zur Identifikation der Methdoe zuweisen. ObjectComponents verwendet den internen Namen zur Verwaltung der automatisierten Elemente. Dieser Name muß dem Namen entsprechen, der der Methode in der Automatisierungsdeklaration mit dem Makro AUTOFUNC zugewiesen wurde.

retType ist ein Datentyp, der den Typ des Rückgabewertes der Methode beschreibt. Wenn, beispielsweise, die Methode einen Longinteger zurückliefert, sollte *retType TAutoLong* sein. Nähere Informationen finden Sie unter Automatisierungsdatentypen.

extName ist der externe, öffentliche Name, den ein Kontroller zur Bezeichnung der Methode verwendet. Dieser String kann lokalisiert werden.

doc ist ein String, der dem Anwender diese Methode beschreibt. Ein Automatisierungskontroller kann diesen String bei OLE abfragen, wenn der Anwender Hilfe anfordert. Auch dieser String kann lokalisiert werden.

help ist eine Kontext-ID für eine .HLP-Datei. Dieser Parameter ist optional und hat nur dann eine Funktion, wenn auch eine .HLP-Datei zur Dokumentierung des Servers registriert wird. Geschieht dies, dann übergibt der Kontroller diese Kontext-ID an das Hilfesystem, das einen Hilfebildschirm mit der Beschreibung der Methode anzeigt, wenn der Benutzer Hilfe anfordert.

id ist ein Zuteilungsbezeichner, den Sie mit einem der EXPOSE_METHOD_ID-Makros auf Wunsch explizit zuweisen können. OLE übergibt diese Zuteilungs-ID zur Identifizierung von Befehlen, die von einem Kontroller angefordert werden. In den Systemheader-Dateien von OLE sind verschiedene Standardwerte für Zuteilungs-IDs definiert. Beispielsweise ist -5 die Standardauswertungsmethode. Standard-Zuteilungs-IDs sind immer negative Zahlen. Von ObjectComponents automatisch vergebene Zuteilungs-IDs werden kleine positive Werte, beginnend mit 1, zugewiesen. Wenn Sie explizite Zuteilungs-IDs für Ihre Anwendungen festlegen möchten, wählen Sie große positive Werte, um Konflikte mit den kleinen positiven Werten, die von ObjectComponents an zugängliche Elemente ohne explizite Zuteilungs-IDs vergeben werden, sicher zu vermeiden.

0 ist die ID der Standardmethode oder Standardeigenschaft. ObjectComponents vergibt niemals automatisch 0 als Zuteilungs-ID. Wenn Sie eine Standardmethode festlegen möchten, müssen Sie 0 selbst zuweisen.

Einem EXPOSE_METHOD-Makro oder EXPOSE_METHOD_ID-Makro muß immer unmittelbar ein Makro für jedes der Argumente der Methode folgen.

Argumenttyp	Deklarationsmakro
Erforderlich	REQUIRED_ARG
Optional	OPTIONAL_ARG
Objekt durch Referenz übergeben	REQBYREF_ARG

Siehe auch: Makros AUTOFUNC, Makro OPTIONAL_ARG, Makro REQBYREF, Makro REQUIRED_ARG

Makros EXPOSE_PROPxxxx ocf/automacr.h

Ein Automatisierungs-Server verwendet EXPOSE_PROPxxxx-Makros in seiner Automatisierungsdefinition (nach DEFINE_AUTOCLASS), um OLE Eigenschaften eines Objekts für die Automatisierung zugänglich zu machen. Eine Eigenschaft ist ein Datum, das nur durch einen Menge von Zugriffsfunktionen (beispielsweise *GetPosition* und *SetPosition*) gelesen oder geschrieben werden kann.

intName ist ein interner Name, den Sie zur Identifikation der Eigenschaft zuweisen. ObjectComponents verwendet interne Namen zur Verwaltung der automatisierten Elemente. Dieser Name muß dem der Methode in der Automatisierungsdeklaration mit dem Makro AUTOPROP zugewiesenen Namen entsprechen.

type ist ein Datentyp, der den Werttyp der Eigenschaft beschreibt. Wenn, beispielsweise, die Eigenschaft ein String ist, dann sollte *type TAutoString* sein. Näheres dazu finden Sie unter Automatisierungsdatentypen.

extName ist der öffentliche Name, den ein Kontroller zum Bezug auf diese Eigenschaft verwendet. Der String kann lokalisiert werden.

doc ist ein String, der dem Anwender diese Eigenschaft beschreibt. Ein Automatisierungskontroller kann diesen String bei OLE anfragen, wenn der Anwender Hilfe anfordert. Der String kann ebenfalls lokalisiert werden.

help ist eine Kontext-ID für eine .HLP-Datei. Dieser Parameter ist optional und hat nur dann eine Funktion, wenn auch eine .HLP-Datei zur Dokumentierung des Servers registriert wird. Geschieht dies, dann übergibt

der Kontroller diese Kontext-ID an das Hilfesystem, das einen Hilfebildschirm mit der Beschreibung der Eigenschaft anzeigt, wenn der Benutzer Hilfe anfordert.

id ist ein Zuteilungsbezeichner, den Sie mit einem der EXPOSE_xxxx_ID-Makros auf Wunsch explizit zuweisen können. OLE übergibt diese Zuteilungs-ID zur Identifizierung von Befehlen, die von einem Kontroller angefordert werden. In den Systemheader-Dateien von OLE sind verschiedene Standardwerte für Zuteilungs-IDs definiert. Beispielsweise ist -5 die Standardauswertungsmethode. Standard-Zuteilungs-IDs sind immer negative Zahlen. Von ObjectComponents automatisch vergebene Zuteilungs-IDs werden kleine positive Werte, beginnend mit 1, zugewiesen. Wenn Sie explizite Zuteilungs-IDs für Ihre Anwendungen festlegen möchten, wählen Sie große positive Werte, um Konflikte mit den kleinen positiven Werten, die von ObjectComponents an zugängliche Elemente ohne explizite Zuteilungs-IDs vergeben werden, sicher zu vermeiden.

0 ist die ID der Standardmethode oder Standardeigenschaft. ObjectComponents vergibt niemals automatisch 0 als Zuteilungs-ID. Wenn Sie eine Standardmethode festlegen möchten, müssen Sie 0 selbst zuweisen.

Makro	Bedeutung
EXPOSE_PROPRW(intName, type, extName, doc, help)	Die Eigenschaft kann gelesen und geschrieben werden.
EXPOSE_PROPRW_ID(id, intName, type, extName, doc, help)	Die Eigenschaft kann gelesen und geschrieben werden. Sie hat die Zuteilungs-ID *id*.
EXPOSE_PROPRO(intName, type, extName, doc, help)	Die Eigenschaft kann nur gelesen werden.
EXPOSE_PROPRO_ID(id, intName, type, extName, doc, help)	Die Eigenschaft kann nur gelesen werden. Sie hat die Zuteilungs-ID *id*.
EXPOSE_PROPWO(intName, type, extName, doc, help)	Die Eigenschaft kann nur geschrieben, aber nicht gelesen werden (selten verwendet).

Makro ocf/automacr.h

EXPOSE_QUIT(extName, docString, helpContext)

Ein Automatisierungs-Server, der die Anwendung selbst als automatisiertes Objekt zugänglich macht, verwendet das Makro EXPOSE_QUIT, um dem Kontroller eine sichere Methode für das Schließen der Anwendung zur Verfügung zu stellen. Die durch dieses Makro implementierte Schließmethode überprüft, ob die Anwendung ursprünglich von OLE für die Automatisierung aufgerufen wurde. Ist das der Fall, entfernt sie den Registriereintrag des aktiven Objekts und schließt den Server. Wenn der Anwender aber den Server schon aufgerufen hatte, bevor der Kontroller eine Verbindung zu ihm herstellte, wird die Schließmethode nicht aktiv, weil die Anwendung weiterlaufen sollte.

EXPOSE-QUIT sollte in der Automatisierungsdefinition des Anwendungsobjekts jeder automatisierten Anwendung stehen (nach DEFINE_AUTOCLASS). EXPOSE_QUIT wird aber nicht in der Automatisierungsdefinition anderer automatisierter Objekte benötigt, die der Server gegebenenfalls erzeugt.

extName ist der externe, öffentliche Name, den ein Kontroller zum Aufrufen dieser Methode verwendet. Der String kann lokalisiert werden.

doc ist ein String, der dem Anwender die Schließmethode beschreibt. Ein Automatisierungskontroller kann diesen String bei OLE abfragen, wenn der Anwender Hilfe anfordert. Auch dieser String kann lokalisiert werden.

help ist eine Kontext-ID für eine .HLP-Datei. Dieser Parameter ist optional und hat nur dann eine Funktion, wenn auch eine .HLP-Datei zur Dokumentierung des Servers registriert wird. Geschieht dies, dann übergibt der Kontroller diese Kontext-ID an das Hilfesystem, das einen Hilfebildschirm mit der Beschreibung der Schließmethode anzeigt, wenn der Benutzer Hilfe anfordert.

Registrierschlüssel extension

Eine Dateinamenserweiterung. Diese Erweiterung ist dann die Standarderweiterung, die Dateinamen im allgemeinen Dialogfenster Datei Öffnen zugewiesen wird. Sie wird darüber hinaus in die Registrierdatenbank des Systems aufgenommen, so daß OLE den richtigen Server für eine Datei anhand der Dateinamenserweiterung finden kann.

extension ist in den Dokumentregistriertabellen von Servern, die Objektverknüpfung und -einbettung unterstützen, gültig. Es ist immer optional. Wenn Sie eine Dateinamenserweiterung registrieren, werden Sie vermutlich auch einen *docfilter* registrieren wollen.

Zum Registrieren einer Dateinamenserweiterung wird das Makro REGDATA verwendet. *extension* wird als erster Parameter übergeben und der Erweiterungs-String als zweiter. In 16-Bit-Windows ist die Dateinamenserweiterung auf drei Zeichen beschränkt.

```
REGDATA(extension, "TXT")
```

Siehe auch: Registrierschlüssel docfilter, Makro REGDATA

Registrierschlüssel filefmt

Registriert einen Namen für das Standard-Dateiformat eines Servers. Dieser String wird in Dialogfenstern angezeigt, in denen der Anwender den Dateityp auswählt.

filefmt ist gültig in Dokument-Registriertabellen für Server, die Objektverknüpfung und -einbettung unterstützen. Es ist immer optional.

Zum Registrieren eines Dateiformats wird das Makro REGDATA verwendet. Dabei wird *filefmt* als erster Parameter übergeben und der Namens-String als zweiter.

Siehe auch: Makro REGDATA

Registrierschlüssel format*n*

Registriert ein Zwischenablageformat, das die Anwendung unterstützt. Eine Anwendung registriert die Formate, die sie in der Zwischenablage ablegen oder aus ihr holen kann. Ein Server kann für verschiedene Dokumenttypen verschiedene Sätze von Zwischenablageformaten registrieren.

Schlüssel für Zwischenablageformate sind in allen Dokument-Registriertabellen gültig. Jede Anwendung, die Objektverknüpfung und -einbettung unterstützt, sollte wenigstens einige Zwischenablageformate registrieren. ObjectComponents unterstützt bis zu acht Formate und verwendet dazu die Schlüssel *format0* bis *format7*. Die Konstante *ocrFormatLimit*, die in ocf/ocreg.h definiert ist, repräsentiert die Höchstzahl der erlaubten Formate (8).

Zum Registrieren von Zwischenablageformaten wird das Makro REGFORMAT verwendet. Der erste Parameter weist dem Format eine Priorität zu. Geben Sie dem von Ihnen bevorzugten Format die höchste Priorität. Programme, die OLE unterstützen, exportieren ihre Daten für gewöhnlich bevorzugt als OLE-Objekte, und daher geben sie *ocrEmbedSource* die Priorität 0. Der zweite Parameter identifiziert ein bestimmtes Dateiformat. Erklärungen zu den anderen Parametern finden Sie in der Beschreibung des Makros REGFORMAT.

```
REGFORMAT(0, octEmbedSource, ocrContent, ocrIStorage, ocrGet)
REGFORMAT(1, ocrMetaFilePict, ocrContent, ocrMfPict, ocrGet)
```

Siehe auch: ocrxxxx-Aspektkonstanten, ocrxxxx-Zwischenablagekonstanten, ocrxxxx-Richtungskonstanten, ocrxxxx-Medienkonstanten, Makro REGFORMAT

Registrierschlüssel handler

Ein vollständiger Pfad, der auf eine Bibliothek zeigt, die die vom Server erzeugten Objekte zeichnen kann.

Ein Pfad für die Bibliothek, die OLE zum Zeichnen von Objekten aufrufen kann, ohne den Server als eigenständigen Prozeß starten zu müssen. In Voreinstellung ist dieser Wert "OLE2.DLL". OLE kann von sich aus Speicherformate wie Metadateien oder Bitmaps darstellen.

Der Schlüssel *handler* ist gültig in Dokument-Registriertabellen von Servern, die Objektverknüpfung und -einbettung unterstützen.Er ist immer optional, sollte aber festgelegt werden, weil OLE sonst die Handler-Bibliothek nicht benutzen kann.

Zum Registrieren eines Handlers wird das Makro REGDATA verwendet. Dabei wird *handler* als erster Parameter übergeben und der Pfad der Handler-DLL im zweiten Parameter. Wenn der Pfad nicht mit einem Laufwerk oder einem Stammverzeichnis beginnt, ermittelt ObjectComponents den vollständigen Pfad ausgehend von dem Laufwerk und Verzeichnis, in dem der Server installiert ist. Wenn beispielsweise der Server sich in C:\MYDIR befindet und der Pfad für den Handler \HELPERS\MYHANDLER ist, geht ObjectComponents von einem vollständigen Pfad C:\MYDIR\HELPERS\MYHANDLER.DLL aus.

Siehe auch: Makro REGDATA

Registrierschlüssel helpdir

Verzeichnis, in dem sich die Online-Hilfe für die Typbibliothek befindet. (Der Name der Hilfedatei wird separat mit dem Schlüssel *typehelp* registriert).

Dieser Schlüssel ist nur in der Anwendungs-Registriertabelle eines Automatisierungs-Servers von Bedeutung. Er ist immer optional. Wenn Sie eine Hilfedatei zu einer Typbibliothek registrieren, ohne zugleich ein Hilfeverzeichnis zu registrieren, geht ObjectComponents automatisch von dem Verzeichnis aus, das für *path* registriert wurde.

Ein Hilfeverzeichnis wird mit dem Makro REGDATA registriert, wobei *helpdir* den ersten Parameter bildet und der Pfad-String den zweiten.

Wenn der Pfad nicht mit einem Laufwerk oder einem Stammverzeichnis beginnt, ermittelt ObjectComponents den vollständigen Pfad ausgehend von dem Laufwerk und Verzeichnis, in dem der Server installiert ist. Wenn beispielsweise der Server sich in C:\MYDIR befindet und der Hilfe-

pfad HELP ist, nimmt ObjectComponents C:\MYDIR\HELP als vollständigen Pfad an.

Siehe auch: Registrierschlüssel path, Registrierschlüssel typehelp

Rückgabekonstanten HR_xxxx ocf/ocfdefs.h

OLE-Systemaufrufe liefern HRESULT-Werte zurück, die manchmal detaillierte Informationen über das Ergebnis des Aufrufs enthalten. Manchmal werden HRESULT-Werte von ObjectComponents an Sie zurückübergeben. Diese Makros vereinfachen das Testen auf einige häufige Ergebnisse. Jedes stellt einen möglichen Rückgabewert dar. Ein vollständiges Verzeichnis von HRESULT-Rückgabewerten finden Sie in der Header-Datei scode.h.

Konstante	Bedeutung
HR_ABORT	Der Vorgang wurde abgebrochen.
HR_FAIL	Es ist ein unspezifizierter Fehler aufgetreten.
HR_FALSE	Eine Aktion wurde nicht in der üblichen Weise ausgeführt, es trat aber kein Fehler auf. Beispielsweise hat eine Aufzählung das Ende der Liste erreicht.
HR_HANDLE	Ein Handle ist ungültig.
HR_INVALIDARG	Ein oder mehr Argumente sind ungültig.
HR_NOERROR	Es ist kein Fehler aufgetreten.
HR_NOINTERFACE	Die angeforderte Schnittstelle wird nicht unterstützt.
HR_NOTIMPL	Der angeforderte Dienst ist nicht implementiert.
HR_OK	Gleiche Bedeutung wie HR_NOERROR.
HR_OUTOFMEMORY	Es steht nicht genügend Speicher zur Verfügung, um den Vorgang zu beenden.
HR_POINTER	Ein Zeiger ist ungültig.

Registrierschlüssel iconindex

Ein null-basierter Index, der darüber Auskunft gibt, welche der Symbole in den Ressourcen des Servers den Objekttyp repräsentieren, den der Server erzeugt.

iconindex wird in den Dokument-Registriertabellen eines Verknüpfungs- und Einbettungs-Servers verwendet. Es ist immer optional.

Um einen Symbolindex zu registrieren, verwenden Sie das Makro REGICON, und übergeben Sie den Indexwert als Parameter.

```
REGICON(1)
```

Siehe auch: Makro REGICON

Registrierschlüssel insertable

Zeigt an, daß die Anwendung ein Server ist und es gestattet, daß ihr Dokument mit anderen Anwendungen verknüpft oder in sie eingebettet wird. Wenn dieser Schlüssel registriert wird, wird der Dokumenttyp in Dialogfenstern angezeigt, die einfügbare Objekte verzeichnen. Der diesem Schlüssel zugewiesene Wert wird ignoriert.

insertable ist in den Dokument-Registriertabellen eines Verknüpfungs-und Einbettungs-Servers gültig.Eine Anwendung muß *insertable* für wenigstens einen Dokumenttyp registrieren, um ein Verknüpfungs- und Einbettungs-Server zu sein. Ein Server muß jedoch nicht alle seine Dokumenttypen einfügbar machen.

Zum Registrieren des Schlüssels *insertable* wird das Makro REGDATA verwendet. *insertable* wird als erster Parameter übergeben und 0 als zweiter.

```
REGDATA(insertable, 0)
```

Siehe auch: Makro REGDATA

Registrierschlüssel language

Setzt die aktuell wirksame Locale-ID außer Kraft. In Voreinstellung erhält der Schlüssel *language* seinen Wert aus der Standard-Spracheinstellung des Systems für den aktuellen Benutzer. Durch die Registrierung einer Spracheinstellung wird ObjectComponents angewiesen, eine bestimmte Sprache für Registrier-Strings zu verwenden, die Sie lokalisiert haben.

Während der Automatisierung wird dieser Wert auf Aufforderung des Automatisierungskontrollers intern zurückgesetzt.

Registrierschlüssel menuname

Ein Kurzname für den Server. Dieser Name wird als Menüeintrag in Container-Programmen angezeigt. Um die Benutzerschnittstelle konsistent zu halten, ist die Länge dieses Namens auf 15 Zeichen beschränkt.

menuname wird in den Dokument-Registriertabellen eines Verknüpfungs- und Einbettungs-Servers benötigt. In anderen Zusammenhängen hat es keine Bedeutung. Der String *menuname* kann lokalisiert werden.

Der Schlüssel *menuname* wird mit dem Makro REGDATA registriert. *menuname* bildet den ersten Parameter und ein Namens-String den zweiten.

```
REGDATA(menuname, "OWL Drawing Pad")
```

Siehe auch: Makro REGDATA

Funktion MostDerived ocf/autodefs.h

const void far* MostDerived(const void far* obj, const typeinfo& src);
Liefert einen Zeiger auf den am weitesten abgeleiteten Klassentyp, der zum angegebenen Objekt paßt, zurück. Dies ist sinnvoll bei der Behandlung polymorpher Objekte. Verwenden Sie diese Funktion, um einen konsistenten Zeiger auf ein Objekt zu erhalten, ohne Rücksicht auf den Zeigertyp, der zum Erreichen des Objekts verwendet wird.

obj ist der Zeiger, dessen am weitesten abgeleiteten Typ Sie bestimmen wollen. *src* enthält Typinformationen über das Objekt *obj*. Der Rückgabewert zeigt auf die am weitesten abgeleitete Klasse, die sich für *obj* finden läßt. Wenn *obj* bereits der am weitesten abgeleitete Typ des Objekts ist, ist der Rückgabewert *obj*.

Verwenden Sie *typeid*, um den Parameter *src* zu erzeugen.

Siehe auch: Funktion DynamicCast, typeid, Klasse typeinfo

Typdefinition ObjectPtr ocf/autodefs.h

typedef void* ObjectPtr;
ObjectPtr ist ein **void***, der auf ein C++ Objekt zeigt.

OC_APPxxxx-Botschaften ocf/ocapp.h

Diese Botschaften werden von ObjectComponents an das Hauptfenster der Anwendung gesandt. Sie benachrichtigen die Anwendungen über Signale und Ereignisse, die vom OLE-System stammen. Die übermittelte Botschaft ist WM_OCEVENT. Die Konstanten in der untenstehenden Tabelle werden im *lParam* der Botschaft übergeben und identifizieren bestimmte Ereignisse. Um den Inhalt des *lParam* herauszufinden, schlagen Sie die passenden Ereignisbehandlungsroutinen nach (wie *EvOcAppBorderSpaceSet* und *EvOcAppMenus*).

Anwendungen, die die ObjectWindows-Bibliothek verwenden, können Ereignisbehandlungsroutinen in ihren Antworttabellen einrichten und verwenden dazu die EV_OC_xxxx-Makros, die in ocfevent.h definiert sind. Weitere Informationen über die Daten, die jede Botschaft enthält, können Sie den Beschreibungen der entsprechenden Ereignisbehandlungsroutinen entnehmen.

Konstanten, die mit OC_APP beginnen, kennzeichnen Ereignisse, die typischerweise im Haupt-Rahmenfenster behandelt werden. Eine weitere Menge von Konstanten, die mit OC_VIEW beginnen, zeigen Ereignisse an, die typischerweise im View-Objekt behandelt werden.

Botschaft	Bedeutung
OC_APPDIALOGHELP	Der Anwender hat die Schaltfläche Hilfe in einem der Standard-Dialogfenster von OLE betätigt.
OC_APPBORDERSPACEREQ	Fragt den Container, ob er dem Server in seinem Hauptfenster Platz für die Rahmendarstellung geben kann.
OC_APPBORDERSPACESET	Bittet den Container, dem Server in seinem Hauptfenster Platz für die Rahmendarstellung zur Verfügung zu stellen.
OC_APPFRAMERECT	Fordert die Koordinaten für das innere Rechteck des Container-Hauptfenster an.
OC_APPINSMENUS	Bittet den Container, sein Menü in der gemeinsamen Menüleiste einzublenden.
OC_APPMENUS	Bittet den Container, die gemeinsame Menüleiste zu installieren.
OC_APPPROCESSMSG	Bittet den Container, Tastenkürzel und andere Botschaften aus der Botschaften-Queue des Servers zu verarbeiten.
OC_APPRESTOREUI	Weist den Container an, seine normalen Menüs und Rahmen wiederherzustellen, da das direkte Bearbeiten im Container-Dokument abgeschlossen ist.
OC_APPSHUTDOWN	Teilt dem Server mit, daß sein letztes verknüpftes oder eingebettetes Objekt geschlossen wird. Wenn der Anwender den Server nicht direkt gestartet hat, kann der Server sich jetzt beenden.
OC_APPSTATUSTEXT	Über gibt während des direkten Bearbeiten im Container-Dokument den Text für die Statuszeile vom Server an den Container.

Siehe auch: OC_VIEWxxxx-Botschaften, WM_OCEVENT-Botschaft

OC_VIEWxxxx-Botschaften ocf/ocview.h

Diese Botschaften werden von ObjectComponents an die Fensterprozedur der Anwendung gesandt. Sie benachrichtigen die Anwendungen über Signale und Ereignisse, die vom OLE-System stammen. Die übermittelte Botschaft ist WM_OCEVENT. Die Konstanten in der untenstehenden Tabelle werden im *wParam* der Botschaft übergeben und identifizieren bestimmte Ereignisse. Um den Inhalt des *lParam* herauszufinden, schlagen Sie die passenden Ereignisbehandlungsroutinen nach (wie *EvOcAppBorderSpaceSet* und *EvOcAppMenus*).

Anwendungen, die die ObjectWindows-Bibliothek verwenden, können Ereignisbehandlungsroutinen in ihren Antworttabellen einrichten und verwenden dazu die EV_OC_xxxx-Makros, die in ocfevent.h definiert sind. Weitere Informationen über die Daten, die jede Botschaft enthält, können Sie den Beschreibungen der entsprechenden Ereignisbehandlungsroutinen entnehmen.

Konstanten, die mit OC_VIEW beginnen, kennzeichnen Ereignisse, die typischerweise im View-Objekt behandelt werden. Eine weitere Menge von

Konstanten, die mit OC_APP beginnen, zeigen Ereignisse an, die typischerweise das Anwendungsobjekt betreffen.

Konstante	Bedeutung
OC_VIEWATTACHWINDOW	Fordert das Server-Fenster auf, sich an sein eigenes Rahmenfenster oder das Containerfenster anzuhängen.
OC_VIEWBORDERSPACEREQ	Fordert einen Rahmenbereich für Werkzeuge zum direkten Bearbeiten im Container-Dokument in der Container-View an.
OC_VIEWBORDERSPACESET	Fordert einen Rahmenbereich für Werkzeuge zum direkten Bearbeiten im Container-Dokument in der Container-View an.
OC_VIEWCLIPDATA	Bittet den Server um Zwischenablage-Daten in einem bestimmten Format.
OC_VIEWCLOSE	Weist den Server an, seine entfernte View zu schließen.
OC_VIEWDRAG	Fordert visuelle Rückmeldung während eines Mausziehvorgangs an.
OC_VIEWDROP	Nimmt ein mit der Maus abgelegtes Objekt entgegen.
OC_VIEWGETPALETTE	Bittet den Server um die Palette, die er zum Zeichnen seines Objekts verwendet.
OC_VIEWGETSCALE	Fordert beim Server Skalierungsinformationen an.
OC_VIEWGETSITERECT	Fordert beim Container das Lagerechteck an.
OC_VIEWINSMENUS	Fordert den Server auf, seine Menüs zum direkten Bearbeiten im Container-Dokument in die Menüleiste einzufügen.
OC_VIEWLOADPART	Fordert den Server auf, sein Dokument zu laden. (Das Dokument des Servers enthält einen Teil).
OC_VIEWOPENDOC	Fragt beim Server die Abmessungen seines geöffneten Dokuments ab.
OC_VIEWPAINT	Fordert den Server auf, eine entfernte View seines Dokuments zu zeichnen.
OC_VIEWPARTINVALID	Zeigt an, daß ein Teil neu gezeichnet werden muß.
OC_VIEWPARTSIZE	Fragt beim Server die Abmessungen seines Objekts ab.
OC_VIEWSAVEPART	Fordert den Server auf, sein Dokument zu speichern (das Dokument des Servers enthält einen Teil).
OC_VIEWSCROLL	Fordert den Client auf, in seiner View einen Bildlauf durchzuführen, weil der Anwender versucht, ein Objekt über die Begrenzung hinaus zu ziehen.
OC_VIEWSETSCALE	Fordert den Server auf, die Skalierung zu behandeln.
OC_VIEWSETSITERECT	Fordert den Container auf, das Lageeechteck zu setzen.
OC_VIEWSHOWTOOLS	Fordert den Server auf, seine Mauspalette für das direkte Bearbeiten im Container-Dokument im Fenster des Containers anzuzeigen.
OC_VIEWTITLE	Fragt den Titel ab, der im Fenster der View angezeigt wird.

Siehe auch: OC_APPxxxx-Konstanten

ocrxxxx-Aspekt-Konstanten ocf/ocreg.h

Diese Konstanten identifizieren Arten der Datendarstellung. Ein Server kann dazu in der Lage sein, das gleiche Objekt auf verschiedene Weise darzustellen, beispielsweise eine vollständige Darstellung des Inhalts anzuzeigen, eine miniaturisierte Wiedergabe des Inhalts oder eine Darstellung des Objekttyps in einem Symbol.

Wenn ein Server ein Datenformat registriert, registriert er auch die Ansicht-Konstanten, die er für jedes Format unterstützt. Die Werte dieser

Konstanten sind Flags und können mit dem bitweisen ODER-Operator (|) kombiniert werden.

Konstante	OLE-Äquivalent	Bedeutung
ocrContent	DVASPECT_CONTENT	Zeigt den vollständigen Inhalt des Objekts in Normalgröße an.
ocrThumbnail	DVASPECT_THUMBNAIL	Zeigt den Inhalt de Objekts verkleinert an.
ocrIcon	DVASPECT_ICON	Zeigt ein Symbol an, das den Objekttyp repräsentiert.
ocrDocPrint	DVASPECT_DOCPRINT	Zeigt das Objekt, wie es auf einem Drucker ausgegeben werden würde.

Siehe auch: ocrxxxx-Konstanten, Makro REGFORMAT, Aufzählungstyp TOcAspect

ocrxxxx-Zwischenablage-Konstanten ocf/ocreg.h

Diese Konstanten identifizieren Standard-Datenformate für Daten, die Anwendungen gemeinsam benutzen können. Sie können sie im Makro REGFORMAT zur Beschreibung der Formate verwenden, die Ihre Dokumente im- und exportieren können.

Konstante	Windows-Formatname	Bedeutung
ocrText	CF_TEXT	Array von Textzeichen
ocrBitmap	CF_BITMAP	Geräteabhängige Bitmap
ocrMetafilePict	CF_METAFILEPICT	Eine Windows-Metadatei, der eine METAFILEPICT-Struktur als Rahmenfunktion dient
ocrSylk	CF_SYLK	Symbolverknüpfungsformat (Symbolic Link Format)
ocrDif	CF_DIF	Datenaustauschformat (Data Interchange Format)
ocrTiff	CF_TIFF	Tag Image File-Format
ocrOemText	CF_OEMTEXT	Text, der Zeichen aus dem Zeichensatz des Originalgeräteherstellers (OEM) enthält (normalerweise ASCII)
ocrDib	CF_DIB	Geräteunabhängige Bitmap
ocrPalette	CF_PALETTE	GDI-Palettenobjekt
ocrPenData	CF_PENDATA	Daten für Pen-Erweiterungen des Betriebssystems
ocrRiff	CF_RIFF	Resource Interchange File-Format (häufig für Multimedia verwendet)
ocrWave	CF_WAVE	Tonschwingungsdatei (verwendet eine Teilmenge des RIFF-Formats)
ocrUnicodeText	CF_UNICODETEXT	Wide-Character Unicode-Text(nur 32-Bit)
ocrEnhMetafile	CF_ENHMETAFILE	Erweiterte Metadatei (nur 32-Bit)
ocrRichText	"Rich Text Format"	RTF-Textformat
ocrEmbedSource	"Embed Source"	OLE-Objekt, das eingebettet werden kann
ocrEmbeddedObject	"Embedded Object"	Bereits eingebettetes OLE-Objekt
ocrLinkSource	"Link Source"	OLE-Objekt, das verknüpft werden kann
ocrObjectDescriptor	"Object Descriptor"	Beschreibende Information über ein einbettungsfähiges OLE-Objekt
ocrLinkSrcDescriptor	"Link Source Descriptor"	Beschreibende Information über ein verknüpfbares OLE-Objekt.

Siehe auch: ocrxxxx-Konstanten, Makro REGFORMAT

ocrxxxx-Richtungskonstanten ocf/ocreg.h

Diese Konstanten identifizieren Richtungen für die Übergabe von Daten. Beispielsweise kann ein Server Bitmaps importieren und exportieren, Metadateien jedoch nur importieren. In diesem Fall verwendet er *ocrGetSet* für das Bitmap-Format und *ocrGet* für Metadateien.

Wenn ein Server ein Datenformat registriert, gibt er auch an, ob er jedes Format importieren oder exportieren kann.

Konstante	Bedeutung
ocrGet	Importiert Daten im angegebenen Format
ocrSet	Exportiert Daten im angegebenen Format
ocrGetSet	Im- und exportiert Daten im angegebenen Format

Siehe auch: ocrxxxx-Konstanten, Makro REGFORMAT

ocrxxxx-Begrenzungskonstanten ocf/ocreg.h

Diese Konstanten legen die Höchstzahl der Aktionsarten und Datenformate fest, die eine Anwendung für jeden einzelnen Dokumenttyp registrieren kann. Aktuell sind beide Begrenzungen auf 8 gesetzt.

Konstante	Bedeutung
ocrVerbLimit	Höchstzahl der Aktionsarten, die ein Server für ein Dokument registrieren kann
ocrFormatLimit	Höchstzahl der Datenformate, die eine Anwendung für ein Dokument registrieren kann

Siehe auch: ocrxxxx-Konstanten

ocrxxxx-Medienkonstanten ocf/ocreg.h

Diese Konstanten identifizieren Kanäle für die Übergabe von Daten. Ein Server ist beispielsweise imstande, eine bestimmte Art Objekt als globales Speicher-Handle, als Dateihandle oder über einen Datenstream zu übergeben.

Wenn ein Server ein Datenformat registriert, dann registriert er dabei auch die Übertragungskanäle, die er für jedes Format unterstützt. Die Werte dieser Konstanten sind Flags und können mit dem bitweisen ODER (|) Operator kombiniert werden.

Konstante	OLE-Äquivalent	Bedeutung
ocrHGlobal	TYMED_HGLOBAL	Handle auf ein globales Speicherobjekt
ocrFile	TYMED_FILE	Handle auf eine Datei
ocrIStream	TYMED_ISTREAM	Stream-Objekt in einer Verbunddatei
ocrIStorage	TYMED_ISTORAGE	Speicherabteil-Objekt in einer Verbunddatei
ocrGDI	TYMED_GDI	GDI-Objekt (wie etwa Bitmap)
ocrMfPict	TYMED_MFPICT	METAFILEPICT-Struktur

Siehe auch: ocrxxxx-Konstanten, Makro REGFORMAT

ocrxxxx-Objektstatuskonstanten ocf/ocreg.h

Diese Konstanten beschreiben, wie ein Objekt sich verhält, wenn es in bestimmten Aspekten dargestellt wird. Registrieren Sie die Optionen für Dokumente mit dem Makro REGSTATUS.

Die Werte dieser Konstanten sind Flags und können mit dem bitweisen ODER (|)-Operator kombiniert werden.

Konstante	Bedeutung
ocrActivateWhenVisible	Gilt nur, wenn *ocrInsideOut* gesetzt ist. Zeigt an, daß das Objekt es vorzieht aktiv zu sein, wann immer es sichtbar ist. Der Container muß dem nicht entsprechen.
ocrCanLinkByOle1	Wird nur in OBJECTDESCRIPTOR verwendet. Zeigt an, daß ein OLE1-Container s ich mit dem Objekt verknüpfen kann.
ocrCantLinkInside	Dieses Objekt sollte nicht zur Verknüpfungsquelle gemacht werden, wenn es eingebettet ist.
ocrInsertNotReplace	Dieses Objekt sollte beim Einfügen in ein Dokument nicht die aktuelle Auswahl ersetzen, sondern neben ihr eingefügt werden.
ocrInsideOut	Dieses Objekt kann aktiviert und bearbeitet werden, ohne Menüs oder Mauspaletten installieren zu müssen. Objekte dieses Typs können parallel mit anderen aktiv sein.
ocrIsLinkObject	Wird von einer OLE 2-Verknüpfung gesetzt, um Kompatibilität zu OLE 1 herzustellen. Das System setzt dieses Bit selbsttätig.
ocrNoSpecialRendering	Gleiche Bedeutung wie *ocrRenderingIsDeviceIndependent*
ocrOnlyIconic	Der Server kann dieses Objekt nur als Symbol sinnvoll darstellen. Die Inhalts-View sieht ebenso aus wie das Symbol.
ocrRecomposeOnResize	Wenn die Container-Lage ihre Größe ändert, möchte der Server das Objekt neu zeichnen. (Vermutlich will der Server in diesem Fall nicht einfach die Skalierung des Objekts ändern).
ocrRenderingIsDeviceIndependent	Das Objekt trifft keine Darstellungsentscheidungen auf der Grundlage des Zielgerätes. Seine Präsentationsdaten sind immer die gleichen.
ocrStatic	Das Objekt ist ein statisches OLE-Objekt und kann nicht bearbeitet werden.

Siehe auch: ocrxxxx-Konstanten, Makro REGSTATUS

ocrxxxx-Verwendungskonstanten ocf/ocreg.h

Diese Konstanten geben darüber Aufschluß, wie ein Server parallel ausgeführte Clients unterstützt. Verwenden Sie sie zum Registrieren des Registrierschlüssels *usage* für einen Server.

Konstante	Bedeutung
ocrSingleUse	Ein Client pro Instanz der Anwendung
ocrMultipeUse	Mehrere Clients pro Instanz der Anwendung
ocrMultipleLocal	Mehrere Clients werden von einem separaten in-proc-Server unterstützt.

Siehe auch: ocrxxxx-Konstanten, Registrierschlüssel usage

ocrxxxx-Aktionsart-Attributkonstanten ocf/ocreg.h

enum ocrVerbAttributes

Diese Container geben dem Container Hinweise über die Einsatzweise einer Aktionsart. Diese Optionen werden für Dokumente mit dem Makro REGVERBOPT registriert.

Die Werte dieser Konstanten sind Flags und können mit dem bitweisen ODER-Operator (|) kombiniert werden.

Konstante	Bedeutung
ocrNeverDirties	Die Aktionsart verändert das Objekt niemals so, daß es gespeichert werden muß.
ocrOnContainerMenu	Die Aktionsart sollte im Containermenü der Objektaktionsarten angezeigt werden, wenn das Objekt aktiv ist. Für die Standardaktionsarten Verbergen, Anzeigen und Öffnen sollte dieses Flag nicht gesetzt sein.

Siehe auch: ocrxxxx-Konstanten, Makro REGVERBOPT

ocrxxxx-Aktionsartmenü-Flags ocf/ocreg.h

Diese Konstanten beschreiben, wie die Aktionsarten eines Servers im Containermenü dargestellt werden sollen. Registrieren Sie diese Optionen für Dokumente mit dem Makro REGVERBOPT.

Die Werte dieser Konstanten sind Flags und können mit dem bitweise ODER-Operator (|) kombiniert werden.

Konstante	Windows-Äquivalent	Bedeutung
ocrGrayed	MF_GRAYED	Stellt die Aktionsart im Menü grau dar. Zugleich wird die Aktionsart damit deaktiviert.
ocrDisabled	MF_DISABLED	Deaktiviert die Aktionsart, so daß der Anwender sie nicht auswählen kann.
ocrChecked	MF_CHECKED	Versieht die Aktionsart mit einer Markierung.
ocrMenuBarBreak	MF_MENUBARBREAK	Positioniert die Aktionsart in einer neuen Spalte und fügt einen vertikalen Trennstrich zwischen den Spalten ein
ocrMenuBreak	MF_MENUBREAK	Positioniert die Aktionsart in einer neuen Spalte, ohne einen Trennstrich einzufügen.

Siehe auch: ocrxxxx-Konstanten, Makro REGVERBOPT

Makro OPTIONAL_ARG ocf/automacr.h

OPTIONAL_ARG(cls, extName, default)
Ein Automatisierungs-Server verwendet dieses Makro in seiner Automatisierungsdefinition (nach DEFINE_AUTOCLASS) zur Beschreibung eines Arguments in einer für den Zugriff zugänglichen Methode.

Nach einem Makro EXPOSE_METHOD oder EXPOSE_METHOD_ID muß eine Liste mit Argumentmakros folgen, eines für jeden Parameter der Methode. Wenn das Argument einen Standardwert hat, wird das Makro OPTIONAL_ARG verwendet.

type ist eine Automatisierungsklasse, die den Datentyp des Arguments beschreibt. Wenn das Argument beispielsweise ein Boolescher Wert ist, sollte *type TAutoBool* sein. Nähere Informationen dazu unter Automatisierungsdatentypen.

extName ist der öffentliche Name, den ein Kontroller verwendet, um sich auf dieses Argument zu beziehen. Dieser String kann lokalisiert werden.

default ist der Standardwert, der zugewiesen wird, wenn der Aufrufer das Argument nicht angibt.

Registrierschlüssel path

Der Pfad, auf dem OLE den Server sucht.

Der Pfad ist für die Anwendungs-Registriertabelle jedes Servers optional. Normalerweise kann die Pfadangabe weggelassen werden, weil Object-Components standardmäßig den tatsächlichen Pfad und den Dateinamen des Servers verzeichnet, wenn dieser sich registriert.

Zum Registrieren des Pfades wird das Makro REGDATA verwendet. Dabei wird *path* als erster Parameter übergeben und der vollständige Pfad-String, einschließlich des .EXE-Namens, als zweiter Parameter.

Siehe auch: Makro REGDATA

Registrierschlüssel permid

Ein String, der die Anwendung ohne Angabe einer Versionsnummer benennt. *permid* entspricht somit *progid*, jedoch ohne Versionsnummer. Er repräsentiert immer die neueste installierte Version einer Klasse.

Der Schlüssel *permid* ist in allen Registriertabellen gültig. Er ist immer optional. Wenn Sie *permid* registrieren, sollten Sie auch *permname* registrieren. Wie *progid* kann auch *permid* nicht lokalisiert werden.

Zum Registrieren von permid wird das Makro REGDATA verwendet. permid stellt den ersten Parameter dar und der ID-String den zweiten.

Siehe auch: Registrierschlüssel permname, Registrierschlüssel progid, Makro REGDATA, Registrierschlüssel version

Registrierschlüssel permname

Ein String, der die Anwendung ohne Angabe einer Versionsnummer beschreibt. *permname* entspricht *description*, jedoch ohne Versionsnummer. Er repräsentiert immer die neueste installierte Version einer Klasse. Ein *permname*-Wert kann bis zu vierzig Zeichen enthalten.

Der Schlüssel *permname* ist in allen Registriertabellen gültig. Er ist immer optional. Wenn Sie *permname* registrieren, sollten Sie auch *permid* registrieren. Der String *permname* sollte lokalisiert werden.

Um *permname* zu registrieren, verwenden Sie das Makro REGDATA und übergeben Sie *permname* als ersten Parameter und den beschreibenden String als zweiten.

Siehe auch: Registrierschlüssel description, Registrierschlüssel permid, Makro REGDATA, Registrierschlüssel version

Registrierschlüssel progid

Registriert einen String, der die Klasse eindeutig identifiziert.

Der String kann bis zu 39 Zeichen enthalten. Das erste Zeichen muß ein Buchstabe sein. Die folgenden Zeichen können Buchstaben, Ziffern oder Punkte sein (jedoch keine Leerzeichen oder andere Trennzeichen). Gemäß Übereinkunft besteht der Wert *progid* aus drei Teilen, die duch Punkt abgetrennt sind. Diese sind der Programmname, ein Objektname und eine Versionsnummer. Der Wert von *progid* kann nicht lokalisiert werden.

Ein *progid*-String ist in jeder Anwendungs-Registriertabelle erforderlich. Zum Registrieren eines *progid*-Wertes wird das Makro REGDATA verwendet. *progid* wird als erster Parameter übergeben und der Bezeichner-String als zweiter.

```
REGDATA(progid, "Drawing Pad.Application.2")
```

Siehe auch: Makro REGDATA

Makro REQUIRED_ARG ocf/automacr.h

REQUIRED_ARG(type, extName);
Ein Automatisierungs-Server verwendet dieses Makro in seiner Automatisierungsdefinition (nach DEFINE_AUTOCLASS), um ein Argument in einer für den Zugriff zugänglich gemachten Methode zu beschreiben.

Nach einem Makro EXPOSE_METHOD oder EXPOSE_METHOD_ID muß eine Liste von Argumentmakros folgen, je eines für jeden Parameter der Methode. Wenn das Argument keinen Standardwert hat und kein Objekt ist, verwenden Sie das Makro REQUIRED_ARG.

type ist eine Automatisierungsklasse, die den Datentyp des Arguments beschreibt. Wenn das Argument beispielsweise ein Boolescher Wert ist, sollte *type TAutoBool* sein. Näheres dazu unter Automatisierungsdatentypen.

extName ist der externe, öffentliche Name, den ein Kontroller verwendet, um sich auf dieses Argument zu beziehen. Der String kann lokalisiert werden.

Klasse TAutoBase ocf/autodefs.h

TAutoBase ist eine Basisklasse zur Ableitung automatisierbarer Objekte. Sie führt nur eine Aktion aus: Wann immer ein Objekt von *TAutoBase* entfernt wird, benachrichtigt der Destruktor OLE, daß das Objekt nicht länger verfügbar ist.

Automatisierte Objekte müssen nicht aus *TAutoBase* abgeleitet werden. Die Ableitung aus *TAutoBase* stellt einfach eine Vorsichtsmaßnahme dar, die nur dann von Bedeutung ist, wenn die Logik des Programms die Zer-

störung des automatisierten Objekts auf nicht automatischem Wege zuläßt, während es noch mit einem OLE-Kontroller verbunden ist.

Wenn Sie *TAutoBase* zur Ableitung von Klassen mit expliziten Klassenbezeichnern verwenden, die nicht den Standardbezeichnungen des Anwendungsmodells entsprechen, müssen Sie darauf achten, das _AUTOCLASS-Makro zu definieren.

Siehe auch: Makro _AUTOCLASS

Öffentlicher Destruktor

Destruktor

virtual ~TAutoBase();
Der virtuelle Destruktor - das einzige Element der Klasse *TAutoBase* - benachrichtigt OLE, wenn das Objekt entfernt wurde. Diese Benachrichtigung ist in solchen Fällen von Bedeutung, in denen das Objekt auf nicht-automatischem Weg - und also ohne daß OLE davon Kenntnis hätte - entfernt werden kann, während es noch mit einem Automatisierungskontroller verbunden ist. Diese Todesanzeige verhindert einen Systemabsturz, der eintreten könnte, wenn OLE versucht, das nicht mehr existente Objekt anzusprechen.

Struktur TAutoBool — ocf/autodefs.h

Verwenden Sie TAutoBool in einer Automatisierungsdefinition, um die Parameter und Rückgabewerte automatisierter Methoden zu beschreiben.

Öffentliches Datenelement

ClassInfo

static TAutoType ClassInfo;
Das Datenelement *ClassInfo* von *TAutoBool* enthält Informationen, die den Booleschen Datentyp identifizieren.

Klasse TAutoCommand — ocf/autodefs.h

TAutoCommand ist eine abstrakte Basisklasse für Automatisierungs-Befehlsobjekte. Immer wenn ein Automatisierungs-Server einen Befehl von einem Automatisierungskontroller erhält, konstruiert er ein Befehlsobjekt. Das Befehlsobjekt empfängt von OLE alle Parameter als VARIANT-Vereinigungsmengen. Der Compiler erzeugt Aufrufe an Konvertierungsfunktionen für Befehlsobjekte, um die passenden C++ Datentypen aus der Vereinigungsmenge zu extrahieren.

Diese Vorgänge laufen intern ab. Normalerweise müssen Sie *TAutoCommand*-Objekte nicht direkt konstruieren oder manipulieren.

Öffentlicher Konstruktor und Destruktor

Konstruktor

TAutoCommand(int attr);
Erzeugt einen Befehl mit den Attributen, die in der Flag-Maske *attr* gesetzt sind. Die Flags sind im Aufzählungstyp *AutoSymFlag* definiert.

Destruktor

virtual ~TAutoCommand();
Entfernt das *TAutoCommand*-Objekt.

Siehe auch: Aufzählungstyp AutoSymFlag

Typdefinitionen

TCommandHook

typedef bool (*TCommandHook)(TAutoCommand& cmdObj);
Beschreibt den Prototyp einer benutzerdefinierten Callback-Funktion, die in der Ausführung von *Invoke*, vor der Ausführung des Automatisierungs-Befehlsobjekts aufgerufen wird. *cmdObj* ist das Objekt, das ausgeführt werden soll. Wenn die Callback-Funktion **false** zurückliefert, führt *Invoke* den Befehl nicht aus.

Siehe auch: TAutoCommand::Invoke, TAutoCommand::SetCommandHook

TErrorMsgHook

typedef const char* (*TErrorMsgHook)(long errCode);
Beschreibt den Prototyp einer benutzerdefinierten Callback-Funktion, die nach *Invoke* aufgerufen wird, um jegliche Art von Fehlercode, die der Befehl eventuell zurückgeliefert hat, zu verarbeiten. *errCode* ist das Statusergebnis. Die Callback-Funktion soll einen String zurückliefern, der dem Anwender den Fehler beschreibt.

Siehe auch: TAutoCommand::LookupError, TAutoCommand::SetErrorMsgHook

Öffentliche Elementfunktionen

ClearFlag

void ClearFlag(int mask);
Löscht alle Flags in *mask*. Die Flags sind im Aufzählungstyp *AutoSymFlag* definiert.

Siehe auch: Aufzählungstyp AutoSymFlag

Execute

virtual void Execute();
Führt den Automatisierungsbefehl aus und ruft dazu das interne C++ Element der automatisierten Klasse auf, zu der der Befehl gehört.

Fail

void Fail(TXAuto::TError);
Löst diejenige Exception aus, die im Parameter angezeigt ist.

Siehe auch: Aufzählungstyp TXAuto::TError

GetSymbol

TAutoSymbol* GetSymbol();
Ermittelt das Symbol, das diesen Befehl generiert.

Invoke

virtual TAutoCommand& Invoke();
Leitet den Vorgang der Ausführung eines Befehls ein. Der Anwender kann das normale Vorgehen außer Kraft setzen, indem er mit dem Makro AUTOINVOKE eine Hook-Prozedur zur Verfügung stellt.

Siehe auch: Makro AUTOINVOKE, TAutoCommand::SetCommandHook

IsPropSet

bool IsPropSet();
Gibt **true** zurück, wenn das Eigenschafts-Flag *asSet* gesetzt ist. Dieses Flag zeigt an, daß der Befehl einer Eigenschaft der automatisierten Klasse einen Wert zuweist und keinen Wert zurückliefert.

LookupError

static const char* LookupError(long errCode);
Übersetzt den Fehlercode einer Funktion in einen Botschafts-String für den Anwender. *errCode* ist ein von *Report* geschickter Funktions-Statuswert. *LookupError* ruft eine Funktion auf, die Sie mit *SetErrorMesgHook* installiert haben. Sie brauchen *LookupError* nicht direkt auzurufen. Wenn Sie eine Hook-Prozedur für Fehlermeldungen eingerichtet haben, wird *LookupError* zum richtigen Zeitpunkt aufgerufen.

Siehe auch: TAutoCommand::Report

Record

virtual int Record(TAutoStack& q);
Zeichnet den Befehl und seine Argumente auf und ruft dazu eine beliebige Hook-Prozedur auf, die der Programmierer über das Makro AUTORECORD in der Automatisierungsdeklaration zur Verfügung gestellt hat.

Die Aufzeichnung wird in der aktuellen Version von ObjectComponents nicht unterstützt.

Siehe auch: Makro AUTORECORD

Report

virtual long Report();
Das Makro AUTOREPORT ruft diese Funktion auf, um den von einem Befehl zurückgelieferten Statuscode in einen Fehlercode zu übersetzen.

Siehe auch: Makro AUTOREPORT; TAutoCommand::SetErrorMsgHook

Return

virtual void Return(TAutoVal& v);
Konvertiert jeden Wert, den der interne C++ Befehl zurückgibt, in eine VARIANT-Vereinigungsmenge. Dieser konvertierte Wert wird an OLE übergeben und bildet den Rückgabewert für den Automatisierungskontroller.

SetCommandHook

static TCommandHook SetCommandHook(TCommandHook callback);
Installiert eine benutzerdefinierte Callback-Funktion des Typs *TCommandHook*, die bei jeder Ausführung des Befehls aufgerufen wird. Die Hook-Prozedur für Befehle ist sinnvoll bei der Überwachung von Automatisierungsaufrufen.

Siehe auch: TCommand::Invoke, TAutoCommand::SetErrorMsgHook, Typdefinition TAutoCommand::TCommandHook

SetErrorMesgHook

static TErrorMsgHook SetErrorMsgHook(TErrorMsgHook callback);
Installiert eine benutzerdefinierte Callback-Funktion des Typs *TErrorMsgHook*, die aufgerufen wird, wenn der Befehl einen Fehlercode zurückliefert.

Siehe auch: TAutoCommand::LookupError, TAutoCommand::Report, TAutoCommand::SetCommandHook, Typdefinition TAutoCommand::TErrorMsgHook

SetFlag

void SetFlag(int mask);
Setzt alle Flags in *mask*. Die Flags sind im Aufzählungstyp *TAutoSymFlag* definiert.

Siehe auch: Aufzählungstyp AutoSymFlag

SetSymbol

void SetSymbol(TAutoSymbol* sym);
Weist dem Befehlsobjekt ein Symbol zu. Das Symbol wird intern gesetzt. Es stammt aus den Tabellen, die von der Automatisierungsdefinition und -deklaration der automatisierten Klasse erzeugt werden.

TestFlag

bool TestFlag(int mask);
Liefert **true** zurück, wenn irgendeines der Flags in *mask* für diesen Befehl gesetzt ist. Die Flags sind im Aufzählungstyp *AutoSymFlag* definiert.

Siehe auch: Aufzählungstyp AutoSymFlag

Undo

virtual TAutoCommand* Undo();
Erzeugt einen Befehl für den UNDO-Stack und ruft dazu jede beliebige Hook-Prozedur auf, die der Programmierer mit dem Makro AUTOUNDO in der Automatisierungsdeklaration zur Verfügung gestellt hat.

Das Rückgängigmachen von Befehlen wird von der aktuellen Version von ObjectComponents nicht unterstützt.

Siehe auch: Makro AUTOUNDO

Validate

virtual bool Validate();
Überprüft die Gültigkeit der Befehlsparameter und führt dazu jede beliebige Prüffunktion aus, die der Programmierer mit dem Makro AUTOVALIDATE in der Automatisierungsdeklaration zur Verfügung gestellt hat.

Siehe auch: Makro AUTOVALIDATE

Geschützte Datenelemente

Attr

int Attr;
Attribut- und Status-Flags. Die Flags sind im Aufzählungstyp *AutoSymFlag* definiert.

Siehe auch: Aufzählungstyp AutoSymFlag

Symbol

TAutoSymbol* Symbol;
Der Symboleintrag, der diesen Befehl erzeugt. OLE übergibt dieses Symbol, um diesen Befehl auszuführen.

Struktur TAutoCurrency — ocf/autodefs.h

TAutoCurrency ist ein Automatisierungsdatentyp, der ObjectComponents hilft, Typprüfung für Elemente einer automatisierten Klasse, die für OLE zugänglich ist, zur Verfügung zu stellen. Verwenden Sie *TAutoCurrency* in einer Automatisierungsdefinition, um Währungswerte zu identifizieren.

Öffentliches Datenelement

ClassInfo

static TAutoType ClassInfo;
Das Element *ClassInfo* von *TAutoCurrency* enthält Informationen, die Daten als Währungswert identifizieren.

Struktur TAutoDate — ocf/autodefs.h

TAutoDate ist ein Automatisierungsdatentyp, der ObjectComponents hilft, Typprüfung für Elemente einer automatisierten Klasse, die für OLE zugänglich ist, zur Verfügung zu stellen. Verwenden Sie *TAutoDate* in einer Automatisierungsdefinition, um Datumswerte, die als Typ **double** gespeichert sind, zu identifizieren.

Öffentliche Datenelemente

ClassInfo

static TAutoType ClassInfo;
Das Datenelement *ClassInfo* von *TAutoDate* enthält Informtionen, die den Datentyp Datum identifizieren.

Date

doubleDate;
Speichert ein Datum als 32-Bit-Wert.

Öffentliche Konstruktoren

Konstruktoren

Form1

TAutoDate();
Erzeugt ein leeres *TAutoDate*-Objekt.

Form2

TAutoDate(double d);
Erzeugt ein *TAutoDate*-Objekt, das zunächst den Wert in *d* enthält; es wird davon ausgegangen, daß *d* ein Datum ist.

Öffentliche Elementfunktion

operator double

operator double();
Liefert den im Datumsfeld von *TAutoDate* gespeicherten Wert zurück.

Struktur TAutoDouble — ocf/autodefs.h

TAutoDouble ist ein Automatisierungsdatentyp, der ObjectComponents hilft, Typprüfung für Elemente einer automatisierten Klasse, die für OLE zugänglich ist, zur Verfügung zu stellen. Verwenden Sie *TAutoDouble* in einer Automatisierungsdefinition, um Werte von Typ **double** zu identifizieren.

Öffentliches Datenelement

ClassInfoClassInfo

static TAutoType ClassInfo;
Das Datenelement *ClassInfo* von *TAutoDouble* enthält Informationen, die den Datentyp **double** identifizieren.

Klasse TAutoEnumerator<> — ocf/autodefs.h

Ein Automatisierungskontroller erzeugt ein *TAutoEnumerator*-Objekt, um Elemente in einer Kollektion aufzuzählen, die einem Automatisierungs-Server gehört. Eine Kollektion kann jede Menge ähnlicher Werte oder Objekte enthalten, die der Server als Gruppe zugänglich machen will. Die Elemente der Kollektion können beispielsweise Zahlen in einem Array sein oder einzelne automatisierte Objekte.

Der Typ, den Sie dem Template übergeben, ist der Werttyp, den die Kollektion enthält. Ist die Kollektion eine Menge von Integerwerten, übergeben Sie **int**. Enthält die Kollektion automatisierte Objekte, übergeben Sie die Platzhalterklasse des Kontrollers (aus *TAutoProxy* abgeleitet).

Zunächst ist ein neu erzeugter Aufzählungstyp leer. Nach dem Aufruf von *Step* enthält der Aufzählungstyp den ersten Wert in der Kollektion. Um den Wert anzuzeigen, rufen Sie *Value* auf, wenn die Kollektion immanente Datentypen, wie **int** oder **float**, enthält, oder *Object*, wenn die Kollektion automatisierte Objekte enthält. *Step, Value* und *Object* sind die wichtigsten Methoden von *TAutoEnumerator*. Die anderen werden im allgemeinen automatisch zum richtigen Zeitpunkt aufgerufen.

Öffentliche Konstruktoren und Destruktor

Konstruktoren>

Form 1

TAutoEnumerator();
Konstruiert ein Aufzählungstyp-Objekt, fügt es aber nicht an eine automatisierte Kollektion an.

Form 2

TAutoEnumerator(const TAutoEnumerator& copy);
Konstruiert ein neues Aufzählungstyp-Objekt, indem es ein vorhandenes kopiert. Beide Aufzählungstypen werden dann an die gleiche Kollektion von Objekte angefügt.

Gleichgültig, auf welche Weise es konstruiert wurde, enthält ein neues *TAutoEnumerator*-Objekt zunächst keinerlei Wert. Rufen Sie immer erst *Step* zum Einlesen des ersten Elements auf, bevor Sie *Value* oder *Object* aufrufen, um das Element anzeigen zu lassen.

Destruktor>

~TAutoEnumerator();
Hebt zunächst die Anbindung des Aufzählungstyps an die Kollektion auf, bevor es ihn zum Entfernen freigibt.

Siehe auch: TAutoEnumerator::StepTAutoEnumerator_Step

Öffentliche Elementfunktionen

Bind

void Bind(TAutoVal& val);
Verbindet den Aufzählungstyp mit dem Kollektionsobjekt *val*. *Bind* wird intern aufgerufen, wenn der Kontroller das Aufzählungstyp-Objekt an eine Methode übergibt, die eine Kollektion zurückliefert.

Siehe auch: TAutoEnumerator::Unbind

Clear>

void Clear();
Leert den Aufzählungstyp, so daß er nicht länger auf ein Element der Kollektion zeigt. Diese Methode wird intern während *Step* aufgerufen.

Siehe auch: TAutoEnumerator::Step

Object>

void Object(TAutoProxy& prx);
Gibt das aktuelle Objekt der Kollektion in *prx* zurück. Verwenden Sie *Object*, wenn die Elemente in der Kollektion automatisierte Objekte sind. Enthält die Kollektion Datenwerte, rufen Sie stattdessen *Value* auf.

Mit *Step* wird der Aufzählungstyp um ein Element weiterbewegt, so daß *Object* das nächste Objekt zurückliefert.

Siehe auch: TAutoEnumerator::Step, TAutoEnumerator::Value

Step>

bool Step();
Bewegt den Aufzählungstyp um einen Schritt weiter, so daß *Value* das nächste Element der Kollektion zurückgibt. Wird *Step* aufgerufen, nachdem der Aufzählungstyp das letzte Element in der Kollektion erreicht hat, gibt es **false** zurück.

Siehe auch: TAutoEnumerator::Object, TAutoEnumerator::Value

Unbind>

void Unbind();
Hebt die Anbindung des Aufzählungstyps an die Kollektion, die er aktuell aufzählt, auf.

Siehe auch: TAutoEnumerator::Bind

Value>

void Value(T& v);
Gibt in *v* das aktuelle Element der Kollektion zurück. Verwenden Sie *Value*, wenn die Kollektion Datenwerte enthält. Enthält die Kollektion Objekte, wird stattdessen *Object* verwendet. Rufen Sie *Step* auf, um den Aufzählungstyp auf das nächste Element weiterzubewegen.

Siehe auch: TAutoEnumerator::Object, TAutoEnumerator::Step

Struktur TAutoFloat ocf/autodefs.h

TAutoFloat ist ein Automatisierungsdatentyp, der ObjectComponents hilft, Typprüfung für Elemente einer automatisierten Klasse, die für OLE zugänglich ist, zur Verfügung zu stellen. Verwenden Sie *TAutoFloat* in einer Automatisierungsdefinition, um Werte von Typ **float** zu identifizieren.

Öffentliches Datenelement

ClassInfo

static TAutoType ClassInfo;
Das Datenelement *ClassInfo* von *TAutoFloat* enthält Informationen, die den Datentyp **float** identifizieren.

Klasse TAutoIterator ocf/autodefs.h

TAutoIterator ist eine abstrakte Basisklasse für Iteratorobjekte. Ein Iterator wird zum Aufzählen einer Kollektion anderer Objekte verwendet. Die Methoden des Iterators führen den Aufrufer Schritt für Schritt durch eine Liste von Objekten und lassen ihn jedes einzelne der Reihe nach untersuchen.

Ein Automatisierungs-Server muß für jedes automatisierte Objekt, das eine Kollektion anderer Objekte repräsentiert, einen Iterator erzeugen. Zum Erzeugen eines Iterators fügt der Server normalerweise ein AUTOITERATOR-Makro in die Automatisierungsdefinition der Kollektionsklasse ein (nach DEFINE_AUTOCLASS).

In den meisten Fällen müssen Sie nicht direkt mit der Iteratorklasse arbeiten, weil das Makro AUTOITERATOR das Objekt für Sie implementiert. Es kann jedoch vorkommen, daß Sie die Klasse direkt definieren müssen, wenn nämlich der Iterator eine besonders komplexe Implementierung benötigt.

Sie können zur Deklaration der Klasse dann trotzdem das Makro AUTOITERATOR_DECLARE verwenden (statt AUTOITERATOR). Dieses Makro erspart Ihnen das manuelle Eintragen aller Standardelemente eines Iteratorobjekts.

TAutoIterator enthält fünf abstrakte Elemente, die jede Ableitungsklasse implementieren muß. Diese fünf Funktionen bilden zusammen eine Standardschnittstelle für Iteratoren in automatisierten Kollektionsobjekten. Sie sind *Init, Test, Step, Return* und *Copy*. Die ersten vier entsprechen Schritten in einer **for**-Schleife, die durch die Kollektion iteriert. (Eine Be-

schreibung dieser Entsprechung finden Sie bei AUTOITERATOR). *Copy* erzeugt eine Kopie des Iterators.

Die Konstruktoren sind geschützt, da *TAutoIterator* nur von einer abgeleiteten Klasse konstruiert werden sollte.

Eine von *TAutoIterator* abgeleitete Klasse implementiert nicht nur die geerbten abstrakten Funktionen, sondern implementiert darüber hinaus typischerweise auch ein oder mehrere Datenelemente, die den aktuellen Zustand des Iterators aufzeichnen. Normalerweise speichert die Statusvariable die Positionen in der Sequenz der aufgezählten Objekte.

TAutoIterator ist ein COM-Objekt und implemetiert die Schnittstelle *IUnknown*.

Öffentliche Elementfunktionen

Copy

virtual TAutoIterator* Copy()=0;
Liefert eine Kopie des Iteratorobjekts zurück. Ihre Implementation sollte die Statusvariablen des Iterators kopieren.

Siehe auch: TAutoIterator::Init, TAutoIterator::Return, TAutoIterator::Step, TAutoIterator::Test

GetSymbol

TAutoSymbol* GetSymbol();
Ermittelt das Automatisierungssymbol, das mit dem Iterator verbunden ist. Normalerweise müssen Sie diese Funkion nicht aufrufen.

Siehe auch: TAutoIterator::SetSymbol

Init

virtual void Init()=0;
Initialisiert alle Statusvariablen im Iterator. Die Hauptaufgabe eines Iterators besteht im schleifenförmigen Durchgehen einer Liste von Objekten, die er einzeln aufzählt. *Init* weist den Iterator an, einen neuen Durchgang durch die Schleife vorzubereiten, Wenn beispielsweise *index* die Statusvariable des Iterators ist, könnte *Init* aussagen

```
index = 0;
```

Siehe auch: TAutoIterator::Copy, TAutoIterator::Return, TAutoIterator::Step, TAutoIterator::Test

IUnknown

operator IUnknown*();
Gibt einen Zeiger auf die OLE-Schnittstelle *IUnknown* des Iterators zurück und ruft *AddRef* für den Schnittstellenzeiger auf. Dieser Operator wird intern aufgerufen, um den Iterator an OLE zurückzuliefern. Normalerweise brauchen Sie ihn nicht direkt aufzurufen.

Return

virtual void Return(TAutoVal& value)=0;
Extrahiert ein Element aus einer Kollektion und gibt eine Referenz darauf im Parameter *value* zurück. Die Hauptaufgabe eines Iterators besteht im schleifenförmigen Durchgehen einer Liste von Objekten, die er einzeln aufzählt. *Return* ist der Befehl, der bei jedem Durchgang durch die Schleife ein anderes Element ermittelt. *Return* kann beispielsweise wie folgt aussehen:

```
value = (Collection->Array)[Index]
```

Dabei ist *value* der Funktionsparameter, *Collection* zeigt auf das einschließende Kollektionsobjekt, *Array* ist ein Element von *Collection,* und *Index* ist die Statusvariable des Iterators.

value ist vom Typ TAutoVal und repräsentiert eine VARIANT-Vereinigungsmenge; in diesem Format werden Werte von OLE übergeben. *TAutoVal* definiert Umwandlungsoperatoren zur Verarbeitung sowohl von Standard-C++ Daten als auch von C++ Strings, *TAutoCurrency*, *TAutoData* und automatisierten C++ Objekten. Die Elemente in einer Kollektion können jedem dieser Typen angehören.

Siehe auch: TAutoIterator::Copy, TAutoIterator::Init, TAutoIterator::Step, TAutoIterator::Test

SetSymbol

void SetSymbol(TAutoSymbol* sym);
Verbindet ein Automatisierungssymbol mit dem Iterator. *SetSymbol* wird intern während der Konstruktion des Iterators aufgerufen. Sie müssen es normalerweise nicht direkt aufrufen.

Siehe auch: TAutoIterator::GetSymbol

Step

virtual void Step()=0;
Setzt den Iterator um einen Schritt weiter, so daß er auf das nächste Element der Kollektion zeigt. Die Hauptaufgabe eines Iterators besteht im schleifenförmigen Durchgehen einer Liste von Objekten, die er einzeln aufzählt. *Step* entspricht der Anweisung i++ in einer **for**-Schleife. Es verändert den Status eines Iterators so, daß er auf das nächste Element zeigt. Wenn beispielsweise die Statusvariable des Iterators *index* heißt, könnte *Step* einfach lauten

```
index++;
```

Siehe auch: TAutoIterator::Copy, TAutoIterator::Init, TAutoIterator::Return, TAutoIterator::Test

Test

virtual bool Test()=0;
Überprüft, ob alle Elemente aufgezählt worden sind. Die Hauptaufgabe eines Iterators besteht im schleifenförmigen Durchgehen einer Liste von Objekten, die er einzeln aufzählt. *Test* gibt **true** zurück, wenn noch weitere Objekte aufzuzählen sind, und **false**, wenn es das Ende der Liste erreicht. Wenn beispielsweise die Statusvariable des Iterators *index* heißt, könnte *Test* so aussehen:

```
return (index >= NUM_ITEMS);
```

Siehe auch: TAutoIterator::Copy, TAutoIterator::Init, TAutoIterator::Return, TAutoIterator::Step

Geschützte Konstruktoren

Konstruktoren

Form1

TAutoIterator (TServedObject& owner);
Konstruiert einen Iterator zur Aufzählung von Elementen der Klasse *owner*. *owner* kann eine beliebige automatisierte Klasse sein.

Form2

TAutoIterator (TAutoIterator& copy);
Konstruiert einen Iterator, indem er eine Kopie eines anderen Iterators erzeugt. Beide Iteratoren zählen die gleiche Kollektion von Objekten auf.

Die Konstruktoren sind geschützt, weil nur abgeleitete Klassen einen *TAutoIterator* konstruieren sollten.

Geschütztes Datenelement

Owner

TServedObject& Owner;
Enthält eine Referenz auf das Objekt der Kollektion, das den Iterator einschließt. *Owner* wird vom Konstruktor initialisiert. Die undokumentierte Klasse *TServedObject* implementiert die Schnittstellen, die ein Client an einem OLE-Objekt zu finden erwartet. ObjectComponents verwendet diese Klasse intern. *Owner* kann ein beliebiges automatisiertes Objekt sein.

Struktur TAutoLong — ocf/autodefs.h

TAutoLong ist ein Automatisierungsdatentyp, der ObjectComponents hilft, Typprüfung für Elemente einer automatisierten Klasse, die für OLE zugänglich ist, zur Verfügung zu stellen. Verwenden Sie *TAutoLong* in einer Automatisierungsdefinition, um Werte von Typ **long** zu identifizieren.

Öffentliches Datenelement

ClassInfo

static TAutoType ClassInfo;
Das Datenelement *ClassInfo* von *TAutoLong* enthält Informationen, die den Datentyp **long** identifizieren.

Klasse TAutoObject<> — ocf/autodefs.h

TAutoObject enthält einen Zeiger auf ein C++ Objekt. *TAutoObject* wandelt die Zeiger auf verschiedene Datentypen entsprechend um, wenn ein Automatisierungsvorgang Typumwandlung erforderlich macht. Es ermittelt darüber hinaus Typinformation über ein Objekt, wenn diese für die Automatisierung benötigt wird. Stellen Sie sich *TAutoObject* als einen intelligenten Zeiger (Smart-Zeiger) vor.

ObjectComponents erzeugt häufig solche Smart-Zeiger für Sie. Normalerweise brauchen Sie *TAutoObject*-Objekte nicht direkt zu bearbeiten.

Öffentliche Konstruktoren

Konstruktoren

Form1

TAutoObject();
Konstruiert ein leeres *TAutoObject*-Objekt, das keinen Zeiger enthält.

Form 2

TAutoObject(T* point);
Konstruiert ein TAutoObject-Objekt, das den Zeiger point enthält.

Form 3

TAutoObject(T& ref);
Konstruiert ein *TAutoObject*-Objekt, das einen Zeiger auf das Objekt *ref* enthält.

Form 4

TAutoObject(IDispatch* dispatch);
Versucht, Typinformationen aus dem Objekt zu lesen, dem die Schnittstelle *IDispatch* gehört. Wenn das gelingt, erzeugt der Konstruktor ein *TAutoObject*-Objekt um einen Zeiger auf das C++ Objekt. Gelingt das Lesen nicht, löst der Konstruktor eine *TXAuto::xTypeMismatch*-Exception aus.

Öffentliche Elementfunktionen

operator *()

T& operator*();
Der Dereferenzierungsoperator gibt eine Referenz auf das Objekt zurück, dessen Zeiger *TAutoObject* enthält.

operator =

Form 1

void operator=(T* point);
Plaziert den Zeiger *point* in dem *TAutoObject*-Objekt.

Form 2

void operator=(T& ref);
Plaziert einen Zeiger auf das Objekt *ref* in dem *TAutoObject*-Objekt.

Form 3

void operator=(IDispatch* dispatch);
Versucht, Typinformationen aus dem Objekt zu lesen, dem die Schnittstelle *IDispatch* gehört. Wenn das gelingt, plaziert der Operator einen Zeiger auf das C++ Objekt in dem *TAutoObject*-Objekt. Gelingt das Lesen nicht, löst der Konstruktor eine *TXAuto::xTypeMismatch*-Exception aus.

Die Zuweisungsoperatoren plazieren einen Zeiger auf ein C++ Objekt in dem *TAutoObject*-Objekt. Sie werden normalerweise verwendet, um das *TAutoObject*-Objekt zu initialisieren, nachdem der Standardkonstruktor es erzeugt hat.

operator T& ()

operator T&();
Gibt eine Referenz auf das Objekt zurück, dessen Zeiger *TAutoObject* enthält.

operator T* ()

operator T*();
Liefert einen Zeiger auf das Objekt zurück, das *TAutoObject* enthält.

TObjectDescriptor()

operator TObjectDescriptor();
Konstruiert ein neues Objektdeskriptor-Objekt auf der Basis des Zeigers, den *TAutoObject* enthält, und liefert dieses Objekt zurück. Dieser Operator wird intern aufgerufen, um Typinformationen zum Konstruieren eines Automatisierungsobjekts zu erhalten.

Geschütztes Datenelement

operator P

T& operator*();
Der Dereferenzierungsoperator liefert eine Referenz aus das Objekt zurück, dessen Zeiger *TAutoObject* enthält.

Klasse TAutoObjectByVal<> ocf/autodefs.h

Basisklasse

TAutoObjectDelete
Ein Automatisierungs-Server verwendet diese Klasse dann, wenn eine automatisierte Methode eine Kopie eines Objekts zurückliefern muß. Normalerweise müssen Sie diese Klasse nicht direkt benutzen, weil die Automatisierungsmakros die korrekten Deklarationen für Sie übernehmen.

Zur Rückgabe eines Objekts dupliziert *TAutoObjectByVal* das Objekt durch Aufrufen seines Kopierkonstruktors. Das Duplikat wird dem Automatisierungskontroller als Rückgabewert eines Automatisierungsbefehls übergeben. *TAutoObjectByVal* hält die Verbindung zu dem duplizierten Objekt so lange aufrecht, bis der Kontroller es freigibt. Dann entfernt es das Objekt durch Aufrufen seines Destruktors.

In jeder anderen Hinsicht ähnelt *TAutoObjectByVal* stark seiner Basisklassse, *TAutoObjectDelete*.

Siehe auch: TAutoObjectDelete

Öffentliches Datenelement

operator =

void operator =(T obj);
Dieser Operator erzeugt eine neues Objekt des Typs *T* durch Kopieren des ursprünglichen Objekts *obj*. Die Kopie wird einem Automatisierungskontroller als Rückgabewert einer automatisierten Methode übergeben. *T* ist der in das Template übergebene Datentyp.

Öffentliche Konstruktoren

Konstruktoren

Form 1

TAutoObjectByVal();
Erzeugt ein leeres *TAutoObjectByVal*-Objekt.

Form 2

TAutoObjectByVal(T obj);
Erzeugt ein *TAutoObjectByVal*-Objekt, das eine Kopie des Objektes *obj* enthält. *T* ist der in das Template übergebene Datentyp.

Klasse TAutoObjectDelete<> ocf/autodefs.h

Basisklasse> Klasse;Basisklasse>

TAutoObject
Ein Automatisierungs-Server verwendet diese Klasse dann, wenn eine automatisierte Methode ein Objekt an einen Automatisierungskontroller zurückliefern muß. Normalerweise müssen Sie diese Klasse nicht direkt benutzen, weil die Automatisierungsmakros die korrekten Deklarationen für Sie übernehmen.

Wie seine Elternklasse *TAutoObject*, besteht *TAutoObjectDelete* zu dem Zweck, einen Zeiger auf ein Objekt aufzunehmen und ihn in der erforderlichen Weise zu konvertieren, wenn das Objekt über Automatisierungsaufrufe vom Server an den Client übergeben wird.Wenn der Automatisierungskontroller die Verarbeitung des automatisierten Objekts beendet hat, informiert *TAutoObjectDelete* - im Unterschied zu *TAutoObject* - das Verbinderobjekt, daß es das automatisierte C++ Objekt seinen Destruktor aufzurufen lassen kann.

Siehe auch: Klasse TAutoObject

Öffentliche Konstruktoren

Konstruktoren

Form 1

TAutoObjectDelete();
Erzeugt ein leeres *TAutoObjectDelete*-Objekt.

Form 2

TAutoObjectDelete(T* p);
Erzeugt ein *TAutoObjectDelete*-Objekt aus einem Zeiger auf ein anderes Objekt.

Form 3

TAutoObjectDelete(T& r);
Erzeugt ein *TAutoObjectDelete*-Objekt aus einer Referenz auf ein anderes Objekt.

Die einzige Aktion der *TAutoObjectDelete*-Konstruktoren besteht darin, ihre Parameter an ihre Elternklasse, *TAutoObject*, zurück zu übergeben.

Öffentliche Elementfunktionen

operator =

Form 1

void operator =(T& r);
Weist *TAutoObjectDelete* an, einen Zeiger auf das Objekt, auf das sich *r* bezieht, anzunehmen.

Form 2

void operator =(T* p);
Weist *TAutoObjectDelete* an, den Zeiger *p* aufzunehmen.

TObjectDescriptor()

operator TObjectDescriptor();
Liefert Typinformation zurück, die das Objekt beschreibt.

Klasse TAutoProxy — ocf/autodefs.h

Ein Automatisierungskontroller leitet Klassen aus *TAutoProxy* ab, um automatisierte OLE-Objekte zu repräsentieren, die er steuern möchte. Um Befehle an ein automatisiertes Objekt zu senden, muß der Kontroller Methoden des Platzhalterobjekts aufrufen, das das Objekt repräsentiert. ObjectComponents verbindet den Platzhalter mit dem Original, so daß das Aufrufen von Elementen des Platzhalters auch Elemente des automatisierten Objekts aufruft.

Ein Platzhalterobjekt muß von *TAutoProxy* erben. In der abgeleiteten Klasse deklariert der Kontroller für jeden Befehl, den er senden möchte, eine Methode. Die deklarierten Methoden müssen mit den Prototypen der gewünschten Befehle übereinstimmen. Zur Implementierung dieser Platzhaltermethoden verwendet der Kontroller drei Makros: AUTONAMES, AUTOARGS und AUTOCALL. Die Makros fügen Code ein, der Aufrufe in die Basisklasse zurückleitet. *TAutoProxy* übergibt die Befehle an OLE.

Normalerweise müssen Sie keinerlei Aufrufe in *TAutoProxy* direkt ausführen. Sie müssen lediglich Ihre Platzhalterklasse aus *TAutoProxy* ableiten und die Methoden mit den Platzhaltermakros implemetieren.

Verwenden Sie das Dienstprogramm AUTOGEN.EXE im Verzeichnis OCTOOLS, um Platzhalterklassen schnell und einfach zu erzeugen. AUTOGEN liest die Typbibliothek des Automatisierungs-Servers und schreibt alle benötigten Header und Quelldateien für Ihre Platzhalterobjekte.

Öffentlicher Destruktor

Destruktor

~TAutoProxy();
Entfernt das *TAutoProxy*-Objekt.

Die Konstruktoren sind geschützt, weil sie nur von abgeleiteten Platzhalterklassen aufgerufen werden sollten.

Öffentliche Elementfunktionen

Bind

Form 1

void Bind(IUnknown* obj);
Bindet das Platzhalterobjekt an einen Server an, der durch einen Zeiger auf seine *IUnknown*-Schnittstelle identifiziert ist. Löst eine *TXOle*-Exception aus, wenn die Anbindung mißlingt.

Form 2

void Bind(IUnknown& obj);
Bindet das Platzhalterobjekt an einen Server an, der durch eine Referenz auf seine *IUnknown*-Schnittstelle identifiziert ist. Löst eine *TXOle*-Exception aus, wenn die Anbindung mißlingt.

Form 3

void Bind(const GUID& guid);
Bindet das Platzhalterobjekt an einen Server an, der durch seinen global eindeutigen Bezeichner (GUID) identifiziert ist. Dies ist die *clsid*, die der Server für Objekte des Typs, den Sie steuern möchten, registriert hat. Löst eine *TXOle*-Exception aus, wenn die Anbindung mißlingt.

Form 4

void Bind(char far* progid);
Bindet das Platzhalterobjekt an einen Server an, der durch seine *progid* identifiziert ist. (Dies ist der GUID, den der Server zur Identifikation der Anwendung selbst registriert hat). Löst eine *TXOle*-Exception aus, wenn die Anbindung mißlingt.

Form 5

void Bind(TAutoVal& val);
Versucht, den Wert in der Vereinigungsmenge *TAutoVal* als Referenz auf ein *IDispatch*-Objekt zu interpretieren, und an *IDispatch* direkt anzubinden. Löst eine *TXAuto*-Exception aus, wenn das Objekt *IDispatch* nicht unterstützt.

Form 6

void Bind(IDispatch* obj);
Akzeptiert *obj* als Server des Platzhalterobjekts.

Form 7

void Bind(IDispatch& obj);
Akzeptiert *obj* als Server des Platzhalterobjekts.

Die Funktion *Bind* versucht, einen Kommunikationskanal zum Automatisierungs-Server zu öffnen, damit Befehle gesandt werden können. Ge-

nauer gesagt fordert *Bind* einen Zeiger auf die *IDispatch*-Schnittstelle des Servers an.

Bind wird intern aufgerufen, wenn das Objekt als Rückgabeobjekt für eine andere Platzhaltermethode übergeben wird. Welche Form von *Bind* zum Einsatz kommt hängt davon ab, welche Informationen zur Identifizierung des Servers zur Verfügung stehen.

Siehe auch: TAutoProxy::Bind

IDispatch&()

operator IDispatch&();
Liefert eine Referenz auf die Schnittstelle *IDispatch* des Servers des Platzhalterobjekts zurück.

IDispatch*();

operator IDispatch*();
Liefert einen Zeiger auf die Schnittstelle *IDispatch* des Servers des Platzhalterobjekts zurück und ruft *AddRef* für den Schnittstellenzeiger auf.

IsBound

bool IsBound();
Gibt **true** zurück, wenn der Server bereits einen Zeiger auf die *IDispatch*-Schnittstelle seines Servers hat, **false**, wenn das nicht der Fall ist.

Lookup

Form 1 **long Lookup(char far* name);**
Ruft den Server auf, um die zum Namen passende ID zu erhalten.

Form 2 **long Lookup(const long id);**
Gibt den in *id* übergebenen Wert zurück.

Form 3 **void Lookup(const char* names, long* ids, unsigned count);**
Schlägt eine Serie von Namen nach und gibt alle ihre IDs zugleich zurück. *names* und *ids* zeigen auf zwei parallele Arrays. *count* gibt die Anzahl der Elemente in beiden Arrays an. Mit nur einem Aufruf an OLE füllt *Lookup* das Array *ids* mit Zahlen, die alle Namen identifizieren.

Wenn ihm der Name eines Befehls oder eines Arguments übergeben wird, ruft *Lookup* den Server auf, um die entsprechenden ID-Werte anzufordern. Obwohl Befehle und Argumente Namen haben, um den Programmierern die Arbeit zu erleichtern, identifiziert OLE sie anhand von

Zahlen. Ein Server muß die ID-Nummer in Erfahrung bringen, um einen Befehl ausführen zu können.

MustBeBound

void MustBeBound();
Löst eine *TXAuto*-Exception aus, wenn das *TAutoProxy*-Objekt keine *IDispatch*-Schnittstelle für seinen Server aufweist. *TAutoProxy* ruft diese Methode intern auf, bevor es Aktionen ausführt, die unterstellen, daß das Objekt bereits an den Server angebunden ist.

SetLang

void SetLang(TLangId lang);
Setzt die Locale-ID, die der Kontroller dem Server zusammen mit jedem Befehl übergibt. Die Locale-ID teilt dem Server mit, welche Sprache der Kontroller verwendet.

Siehe auch: Locale-ID, TAutoStack

Unbind

void Unbind();
Verringert den Referenzzähler des Servers des Platzhalterobjekts und löscht die internen Referenzen auf den Server.

Siehe auch: TAutoProxy::Bind

Geschützter Konstruktor

Konstruktor

TAutoProxy(TLangId lang);
Konstruiert ein *TAutoProxy*-Objekt und stellt es auf die Sprache ein, die in der Locale-ID *lang* identifiziert ist.

Siehe auch: Locale-ID

Geschützte Elementfunktion

Invoke

TAutoVal& Invoke(int attr, TAutoProxyArgs& args, long* ids, unsigned named=0);
Sendet einen Befehl an den Automatisierungs-Server. *Invoke* wird von den AUTOCALL-Makros aufgerufen.

attr beschreibt den Typ des ausgegebenen Befehls und kann eine Kombination der Werte des Aufzählungstyps *AutoCallFlag* sein.

Das Objekt *args* enthält alle Werte, die dem Befehl als Argumente übergeben werden.

ids zeigt auf ein Array von ID-Werten, die den Befehl und die Argumente identifizieren. Es sollte einen ID-Wert für jedes Element im Array *args* geben.

names gibt an, wie viele Argumente in *args* durch Namen identifiziert sind.

Siehe auch: Aufzählungstyp AutoCallFlag, Makros AUTOCALL_xxxx

Struktur TAutoShort ocf/autodefs.h

TAutoShort ist ein Automatisierungsdatentyp, der ObjectComponents hilft, Typprüfung für Elemente einer automatisierten Klasse, die für OLE zugänglich ist, zur Verfügung zu stellen. Verwenden Sie *TAutoShort* in einer Automatisierungsdefinition, um Werte von Typ **short** zu identifizieren.

Öffentliches Datenelement

ClassInfo

static TAutoType ClassInfo;
Das Datenelement *ClassInfo* von *TAutoShort* enthält Informationen, die den Datentyp **short** identifizieren.

Klasse TAutoStack ocf/autodefs.h

TAutoStack verarbeitet den Befehls-Stack, den ein Automatisierungskontroller durch OLE an einen Automatisierungs-Server sendet. Der Befehls-Stack enthält eine Zuteilungs-ID, die einen bestimmten Befehl identifiziert und eine Menge von VARIANT-Vereinigungsmengen, die alle Argumente enthalten, die zur Ausführung des Befehls erforderlich sind.

ObjectComponents interpretiert die Zuteilungs-ID und extrahiert den richtigen C++ Wert aus jeder Vereinigungsmenge. Es erzeugt ein Befehlsobjekt (*TAutoCommand*) und ruft die *Execute*-Methode des Befehls auf. *TAutoCommand* ruft im Gegenzug die Methoden auf, die Sie zugänglich gemacht haben, indem Sie sie in Ihren automatisierten Klassen deklariert und definiert haben.

Der Stack enthält außerdem eine Locale-ID, die die im Befehl verwendete Sprache identifiziert. ObjectComponents berücksichtigt bei der Interpreta-

tion von Strings, die es aus dem Stack extrahiert, die Localekennung. Wenn Sie Lokalisierungsressourcen zur Verfügung gestellt haben, übersetzt ObjectComponents für Sie in die gewünschte Sprache.

Normalerweise müssen Sie mit *TAutoStack* nicht direkt arbeiten. ObjectComponents übergibt automatisch einen Stack in das richtige Befehlsobjekt. Die Befehlsobjekte werden von den Automatisierungsdeklarations-Makros erzeugt.

Siehe auch: Klasse TAutoCommand

Öffentlicher Konstruktor und Destruktor

Konstruktor

TAutoStack(TServedObject& owner, VARIANT far* stack, TLocaleId locale, int argcount, int namedcount, long far* map);

Der Konstruktor wird nur intern aufgerufen. Es sollte nicht erforderlich sein, daß Sie Ihren eigenen Stack konstruieren.

- *owner* ist das automatisierte Objekt, an das der Befehl gerichtet ist.
- *stack* zeigt auf eine Serie zusammenhängender Vereinigungsmengen vom Typ VARIANT. Die Vereinigungsmengen enthalten Werte oder Objektreferenzen, die in Automatisierungsbefehlen übergeben werden.
- *locale* ist eine Locale-ID, die die vom Kontroller benutzte Sprache beschreibt.
- *argcount* gibt an, wie viele Argumente im Stack auf die Zuteilungs-ID folgen.
- *namedcount* gibt an, wie viele der Argumente mit ihren Namen übergeben wurden. Ein Kontroller kann Argumente in beliebiger Reihenfolge übergeben und sogar optionale Argumente auslassen, wenn er die von ihm übergebenen Argumente explizit mit den Namen identifiziert, die ihnen der Server gegeben hat.

Wenn *namedcount* größer 0 ist, zeigt *map* auf ein Array von ID-Werten, die den vom Konstruktor übergebenen Argumentnamen entsprechen.

- *map* ist eine Tabelle zur Übersetzung benannter Argument-IDs in Argumentpositionen.

Destruktor

~TAutoStack();

Entfernt das *TAutoStack*-Objekt.

Öffentliche Elementfunktion

operator[]

TAutoVal& operator [](int index);
Extrahiert einzelne Argumente aus dem Befehls-Stack zur Verwendung als C++ Funktionsargumente. *index* ist ein nullbasierter Index in die Argumentliste des Befehls, deren Reihenfolge der Festlegung durch das entsprechende EXPOSE-Makro der Automatisierungsdefinition entspricht. Dieser Operator wird von den in der Automatisierungsdeklaration generierten Befehlsobjekten aufgerufen.

Liegt *index* außerhalb des Gültigkeitsbereichs, löst der Operator eine *TXAuto::xNoArgSymbol*-Exception aus.

Öffentliche Datenelemente

ArgCount

const int ArgCount;
Enthält die Anzahl der auf dem Befehls-Stack übergebenen Argumente (benannt oder unbenannt).

ArgSymbolCount

int ArgSymbolCount;
Enthält die Anzahl der für die Automation zugänglich gemachten Befehlsargumente.

CurrentArg

int CurrentArg;
Indiziert das aktuelle Argument, während ObjectComponents die Argumente auf dem Stack eins nach dem anderen abarbeitet. Wenn *CurrentArg ArgCount* erreicht, wurden alle Argumente verarbeitet.

LangId

TLangId LangId;
Enthält eine Zahl, die die Sprache identifiziert, die der Kontroller zum Senden von Befehlen verwendet.

Owner

TServedObject& Owner;
Bezieht sich auf das automatisierte Objekt, das den Befehl auf dem Stack verarbeitet.

Symbol

int ArgSymbolCount;
Enthält die Anzahl der Befehlsargumente, die für die Automatisierung zugänglich sind.

Konstante

SetValue

TAutoStack::SetValue
SetValue ist eine vordefinierte Standard-Zuteilungs-ID. Die Zuteilungs-ID ist eine Zahl zur Identifizierung eines bestimmten Befehls, den ein automatisiertes Objekt ausführen kann. In ObjectComponents werden nur zwei Standard-Zuteilungs-IDs verwendet, 0 für die Standardaktion eines Objekts und -3 für einen Befehl, der den Wert einer Eigenschaft setzt. *SetValue* ist -3.

Siehe auch: Klasse TAutoVal, TLocaleId

Struktur TAutoString — ocf/autodefs.h

Ein Automatisierungs-Server verwendet *TAutoString*, um C-Stringtypen in einer Automatisierungsdefinition zu beschreiben. Die Elementfunktionen der Struktur *TAutoString* erleichtern das Kopieren und Zuweisen von Stringwerten. Beim Übergeben der Strings zwischen Servern und Kontrollern werden dabei nur minimale Realloziierungen des Speichers erforderlich.

Bei C++ *string*-Objekten brauchen Sie *TAutoString* nicht zu verwenden. Näheres dazu finden Sie unter Automatisierungsdatentypen.

TAutoString arbeitet am besten mit Stringwerten vom Typ **const**. Wenn ihm ein nicht-konstanter String übergeben wird, muß *TAutoString* eine interne Kopie anlegen. Ist der String aber ein **const**-Typ, weiß *TAutoString*, daß der Wert sich nicht ändern wird, und läßt den Kopiervorgang aus. Das führt zu einer erheblich höheren Verarbeitungsgeschwindigkeit.

Öffentliche Konstruktoren und Destruktor

Konstruktoren

Form 1

TAutoString(const string& s);
Erzeugt einen *TAutoString* und weist ihm den in einem C++ *string*-Objekt enthaltenen String zu.

Form 2

TAutoString(const TAutoString& copy);
Erzeugt einen neuen *TAutoString*, der den gleichen Stringwert enthält wie das Objekt *copy*.

Form 3

TAutoString(TAutoVal& val);
Initialisiert das neue Objekt mit dem Wert in einer *TAutoVal*-Vereinigungsmenge. *TAutoVal* repräsentiert den Datentyp VARIANT, den OLE zur Übergabe von Werten zwischen zwei Anwendungen verwendet. Es ist eine Vereinigungsmenge vieler Typen. Dieser Konstruktor extrahiert den Wert aus der Vereinigungsmenge als String.

Form 4

TAutoString(const char far* str);
Initialisiert das neue Objekt mit dem Wert in einem C-String vom Typ **const**.

Form 5

TAutoString(BSTR s, bool loan);
Initialisiert das neue Objekt mit dem Wert eines Strings im BASIC-Format, also einem String, dem seine Länge vorangestellt ist, und der nicht mit Null abgeschlossen ist. Dieses Format verwendet OLE zur Übergabe von Strings. *loan* wird auf **true** gesetzt, wenn das *TAutoString*-Objekt den BSTR besitzt und auf **false**, wenn es den BSTR lediglich referenziert.

Destruktor

~TAutoString();
TAutoString besitzt einen Referenzzähler für die Stringobjekte, die es enthält. Der Destruktor verringert diesen Referenzzähler.

Öffentliche Elementfunktionen

int()

operator int()
Liefert die Länge des Stringwertes zurück (so, wie *strlen* sie berechnen würde).

operator =

Form 1

TAutoString& operator =(const char far* str);
Akzeptiert einen **const**-String in C-Schreibweise als neuen Wert von *TAutoString*.

Form 2

TAutoString& operator =(char* s);
Akzeptiert einen nicht-**const**-String in C-Schreibweise als neuen Wert von *TAutoString*. Da der String nicht konstant ist, muß *TAutoString* eine Kopie des Strings für sich selbst erzeugen. Dadurch wird Form 2 erheblich langsamer ausgeführt als Form 1. Versuchen Sie, nach Möglichkeit **const**-Strings zu übergeben.

Form 3

TAutoString& operator =(const TAutoString& copy);
Setzt den Wert des *TAutoString*-Objekts so, daß er eine String-Kopie aus einem anderen *TAutoString*-Objekt darstellt.

char*()

operator char*()
Gibt den String-Wert des Objekts in Form eines nicht-**const** Strings in C-Schreibweise zurück. Zu diesem Zweck muß *TAutoString* eine Kopie des Strings anlegen. Es ist schneller, einen **const char*** zuzuweisen, wann immer das möglich ist.

const char far*()

operator const char far*();
Liefert den String-Wert des Objekts in Form eines **const**-Strings in C-Schreibweise zurück.

Öffentliches Datenelement

ClassInfo

static TAutoType ClassInfo;
Diese statische Struktur enthält eine Zahl, die den Datentyp als String identifiziert. Alle Automatisierungsdatentypen enthalten einen entsprechenden statischen Bezeichner, so daß ObjectComponents jeden einzelnen von ihnen abfragen kann, um ihre Identität festzustellen.

Struktur TAutoType — ocf/autodefs.h

Die Struktur *TAutoType* ist ein statisches Datenelement aller Klassen von Automatisierungsdatentypen, wie *TAutoBool* und *TAutoString*. *TAutoType* macht alle diese Datentypen selbstbeschreibend. Das ist eine entscheidende Eigenschaft für den Umgang mit den VARIANT-Vereinigungsmengen, die OLE verwendet, um Werte während der Automatisierung zu übergeben. Da alle Automatisierungsdatentypen aus *TAutoType* abgeleitet sind, kann ObjectComponents Werte beliebigen Typs mit dem gleichen Code verarbeiten. Und weil *TAutoType* selbstbeschreibend ist, kann ObjectComponents jederzeit den tatsächlichen Typ jedes beliebigen Elements feststellen.

Normalerweise müssen Sie nicht mit *TAutoType* direkt arbeiten, sondern nur mit den Automatisierungstypen, die daraus abgeleitet sind.

Öffentliche Elementfunktion

GetType

short getType();
Liefert eine Ganzzahl zurück, die einen bestimmten Datentyp identifiziert. Die Bezeichner sind im Aufzählungstyp *AutoDataType* definiert.

Siehe auch: Aufzählungstyp AutoDataType

Klasse TAutoVal ocf/autodefs.h

TAutoVal dupliziert den VARIANT-Typ, den OLE verwendet, um Werte zwischen einem Automatisierungs-Server und einem Automatisierungskontroller zu übergeben. Es fügt außerdem Zugriffsmethoden hinzu, um den Wert in dem VARIANT zu ermitteln. Ein VARIANT kann in den Typ *TAutoVal* umgewandelt werden und umgekehrt.

Ein VARIANT ist eine große Vereinigungsmenge mit Feldern vieler verschiedener Datentypen. Eine große Menge aus überladenen Zuweisungsoperatoren gestattet es, viele verschiedenartige Werte in einem *TAutoVal*-Objekt zu speichern. Jeder Zuweisungsoperator zeichnet darüber hinaus intern eine Zahl auf, die den gerade entgegengenommenen Werttyp identifiziert. Eine ähnliche Menge von Konvertierungs-Operatoren erlaubt die Umwandlung des Wertes im Objekt in verschiedene andere Wertetypen. Ob eine bestimmte Umwandlung erfolgreich ist, hängt vom Typ des Wertes im Objekt ab. Beispielsweise kann ein *string* nicht in ein anderes Objekt umgewandelt werden. Wenn die Umwandlung mißlingt, löst *TAutoVal* eine Exception vom Typ *TXAuto::xConversionFailure* aus.

ObjectComponents behandelt die Daten, die zwischen einem Automatisierungs-Server und einem Automatisierungskontroller übergeben werden als einen Stack von Vereinigungsmengen. Der Stack ist *TAutoStack*, und die Elemente auf dem Stack sind *TAutoVal*. Da Server und Controller unabhängig voneinander aufgebaut sind und verschiedene Programmiersprachen verwenden können, können die zwischen ihnen ausgetauschten Daten nicht einem immanenten Typ entsprechen. Befehlsbezeichner und Argumentwerte werden als VARIANTs übergeben. Der Empfänger eines VARIANT-Wertes muß sich auf den Kontext eines Elementes stützen, um festzustellen, welchem Typ der Wert angehören soll. Wenn die Anwendung beispielsweise eine Zuteilungs-ID für einen Befehl empfängt, der zwei Integerargumente erwartet, extrahiert sie Integerwerte aus den zwei folgenden VARIANTs.

Öffentliche Elementfunktionen

operator =

```
void operator=(int i);
void operator=(int far* p);
void operator=(long i);
void operator=(long far* p);
void operator=(unsigned long i);
void operator=(unsigned long far* p);
void operator=(short i);
void operator=(short far* p);
void operator=(float i);
void operator=(float far* p);
void operator=(double i);
void operator=(double far* p);
void operator=(TBool i);
void operator=(TBool far* p);
void operator=(const char far* s);
void operator=(string s);
void operator=(TAutoString s);
void operator=(TAutoCurrency i);
void far* operator=(TAutoCurrency far* p);
void operator=(TAutoDate i);
void far* operator=(TAutoDate far* i);
void operator=(TAutoVoid);
void operator=(IDispatch* ifc);
void operator=(IUnknown* ifc);
void operator=(TObjectDescriptor od);
```

Zuweisungsoperatoren initialisieren *TAutoVal*, indem sie in dem Objekt sowohl den zugewiesenen Wert als auch eine ID plazieren, die den Typ des zugewiesenen Wertes anzeigt.

Diese Tabelle beschreibt die Datentypen, die nicht Standard-C-Typen sind.

Typ	Beschreibung
IDispatch	Eine Klasse, die von ObjectComponents intern verwendet wird, um die OLE Standardschnittstelle *IDispatch* zu implementieren, die von automatisierbaren Objekten unterstützt wird.

IUnknown	Eine Klasse, die ObjectComponents intern zur Implementierung der OLE Standardschnittstelle *IUnknown* verwendet, die von allen OLE-Objekten unterstützt wird.
string	C++ String-Objekt.
TAutoCurrency	Ein Automatisierungsdatentyp, der einen Währungswert enthält.
TAutoString	Ein Automatisierungsdatentyp, der einen String-Wert in C-Schreibweise enthält.
TAutoVoid	Ein Automatisierungdatentyp, der einen Rückgabewert **void** repräsentiert.
TObjectDescriptor	Eine Klasse, die ObjectComponents intern verwendet, um Informationen über ein OLE-Objekt aufzunehmen.

Siehe auch: Automatisierungsdatentypen

TBool far*()

operator TBool far*();
Gibt einen Zeiger auf einen Booleschen Wert zurück.

TBool()

operator TBool();
Gibt den Wert in dem Objekt als Booleschen Wert zurück.

Clear

void Clear();
Löscht den in dem Objekt gespeicherten Wert und läßt das leere Objekt zurück. Diese Methode kann nicht für Objekte aufgerufen werden, die von *TAutoStack* verwaltet werden.

Siehe auch: Klasse TAutoStack

Copy

void Copy(const TAutoVal& copy);
Kopiert das *TAutoVal*-Objekt in *copy*. Weist, falls erforderlich, intelligent Platz für einen String zu und ruft *AddRef* auf, wenn der Wert in der Vereinigungsmenge ein OLE-Objekt ist.

double far*()

operator double far*();
Liefert einen Zeiger auf einen **double**-Wert zurück.

double()

operator double();
Gibt den Wert im Objekt als **double**-Wert zurück.

float far*()

operator float far*();
Liefert einen Zeiger auf einen Gleitkommawert zurück.

float()

operator float();
Gibt den Wert im Objekt als Gleitkommawert zurück.

GetDataType

int GetDataType();
Gibt eine Ganzzahl zurück, die den der Vereinigungsmenge zugewiesenen Werttyp identifiziert.

IDispatch&()

operator IDispatch&();
Extrahiert eine *IDispatch*-Schnittstelle aus dem Wert in dem *TAutoVal*-Objekt. *IDispatch* ist die OLE-Standardschnittstelle, die von automatisierbaren Objekten unterstützt wird. Diese Methode ruft nicht *AddRef* für die *IDispatch*-Schnittstelle auf.

Siehe auch: TAutoVal::IDispatch*()

IDispatch*()

operator IDispatch*();
Extrahiert eine *IDispatch*-Schnittstelle aus dem Wert in dem *TAutoVal*-Objekt. *IDispatch* ist die OLE-Standardschnittstelle, die von automatisierbaren Objekten unterstützt wird. Diese Funktion ruft auch die Methode *AddRef* der Schnittstelle auf.

Siehe auch: TAutoVal::IDispatch&()

int far*()

operator int far*();
Gibt einen Zeiger auf einen **int** zurück.

int()

operator int();
Gibt den Wert im Objekt als Ganzzahl zurück.

IsRef

bool IsRef();
Liefert **true** zurück, wenn der der Vereinigungsmenge zugewiesene Wert eine Referenz auf einen Wert ist.

IUnknown&()

operator IUnknown&();
Extrahiert aus dem Wert in dem *TAutoVal*-Objekt eine *IUnknown*-Schnittstelle. *IUnknown* ist die OLE-Standardschnittstelle, die von allen Objekten unterstützt wird. Diese Methode ruft nicht *AddRef* für die *IUnknown*-Schnittstelle auf.

Siehe auch: TAutoVal::IUnknown*()

IUnknown*()

operator IUnknown*();
Extrahiert aus dem Wert in dem *TAutoVal*-Objekt eine *IUnknown*-Schnittstelle. *IUnknown* ist die OLE-Standardschnittstelle, die von allen Objekten unterstützt wird. Diese Methode ruft *AddRef* für die *IUnknown*-Schnittstelle auf.

Siehe auch TAutoVal::IUnknown&()

long far*()

operator long far*();
Gibt einen Zeiger auf einen Longinteger zurück.

long()

operator long();
Gibt den Wert im Objekt als Longinteger zurück.

short far*()

operator short far*();
Liefert einen Zeiger auf einen Shortinteger zurück.

short()

operator short();
Gibt den Wert im Objekt als Shortinteger zurück.

string()

operator string();
Gibt den Wert im Objekt als C++ *string*-Objekt zurück.

far*()

operator int far*();
Liefert einen Zeiger auf einen **int** zurück.

TAutoCurrency()

operator TAutoCurrency();
Gibt den Wert im Objekt als Währungswert zurück.

Siehe auch: Klasse TAutoCurrency

TAutoCurrency far*()

operator TAutoCurrency far*()
Liefert einen Zeiger auf einen Währungswert zurück.

Siehe auch: TAutoCurrency

TAutoDate far*()

operator TAutoDate far*();
Liefert einen Zeiger auf einen Datumswert zurück.

Siehe auch: Klasse TAutoDate

TAutoDate()

operator TAutoDate();
Gibt den Wert im Objekt als Datumswert zurück.

Siehe auch: Klasse TAutoDate

TUString*()

operator TUString*();
Gibt den Wert im Objekt als *TUString*-Objekt zurück. *TUString* ist eine Vereinigungsmenge verschiedener Stringrepräsentationen mit Referenzzähler. Es wird intern von ObjectComponents zur Implemetierung von *TAutoString* verwendet.

unsigned long()

operator unsigned long();
Liefert den Wert im Objekt als vorzeichenlosen Longinteger zurück.

unsigned long far*()

operator unsigned long far*();
Gibt einen Zeiger auf einen Longinteger zurück. (*TAutoVal* unterscheidet nicht zwischen **long** und **unsigned long**).

Siehe auch: Klasse TAutoStack.

Struktur TAutoVoid — ocf/autodefs.h

TAutoVoid ist ein Automatisierungsdatentyp wie *TAutoShort* und *TAutoBool*. Verwenden Sie ihn in Automatisierungsdefinitionen zur Beschreibung von Funktionen, die keinen Wert zurückliefern.

Der Zweck der Struktur ist, die Zuweisung von **void** zu einem *TAutoVal* zu implementieren.

Siehe auch: Struktur TAutoVal

Öffentliches Datenelement

ClassInfo

static TAutoType ClassInfo;
Wie bei allen Automatisierungsdatentypen enthält das Datenelement *ClassInfo* einen Wert, der einen Datentyp identifiziert, in diesem Fall **void**.

Siehe auch: Struktur TAutoType

Typdefinition TComponentFactory — ocf/ocreg.h

typedef IUnknown* (*TComponentFactory)(IUnknown* outer, uint32 options, uint32 id = 0);
TComponentFactory ist eine Typdefinition für eine Callback-Funktion.

outer zeigt auf die Schnittstelle *IUnknown* einen externen OLE-Objekts. Die Anwendung ist aufgefodert, sich in diesem Objekt zu versammeln (Aggregation). Ist *outer* gleich 0, dann ist das neue Objekt entweder unabhängig, oder es wird zum äußeren Objekt einer solchen Aggregation.

option enthält Bitflags, die den Ausführungszustand der Anwendung anzeigen. Zum Testen der Flags werden die *TOcAppMode* Aufzählungskonstanten verwendet.

id ist eine Zahl, die ObjectComponents vergibt, um einen bestimmten Objekttyp, den die Anwendung erzeugen kann, zu identifizieren. Wenn *id* 0 ist, wird die Anwendung aufgefordert, sich selbst zu erzeugen. Um bestimmte Dokumenttypen anzufordern, übergibt ObjectComponents die Dokument-Template-ID.

Der Rückgabewert ist ein Zeiger auf die Schnittstelle *IUnknown* eines beliebigen von der Callback-Funktion erzeugten Objekts, sei es die Anwendung selbst oder eines ihrer Objekte. Während der Aggregation wird der Rückgabewert zum inneren *IUnknown*-Zeiger in einem anderen Objekt (*IUnknown* ist ein OLE-Standardtyp, der in compobj.h deklariert ist).

Ein Callback vom Typ *TComponentFactory* wird dem Konstruktor eines Registrar-Objekts (entweder *TOcRegistrar* für eine Verknüpfungs- und Einbettungsanwendung oder *TRegistrar* für eine Anwendung, die nur Automatisierung unterstützt).

Siehe auch: Aufzählungstyp TOcAppMode, Klasse TOcRegistrar, Klasse TRegistrar

Typdefinition TLocaleId — ocf/autodefs.h

typedef unsigned long TLocaleId;

Eine Locale-ID ist ein 32-Bit-Wert, der eine Sprache identifiziert. Die niedrige Hälfte des Wertes ist eine 16-Bit Sprachkennung. In der aktuellen OLE-Definition ist das obere Wort reserviert, im Ergebnis ist also eine Locale-ID eine 32-Bit-Sprachkennung.

Windows verwendet Locale-IDs, um die Standardsprache des Systems einzustellen. ObjectComponents verwendet Locale-Ids in der Automatisierung. Ein Automatisierungskontroller übergibt dem Server mit jedem Befehl eine Locale-Id. Vom Server wird erwartet, daß er die von ihm empfangenen Befehle als Strings in der angegebenen Sprache interpretiert.

Es gibt zwei vordefinierte Locale-Einstellungen für das System in der Header-Datei olenls.h.

Konstante	Bedeutung
LOCALE_SYSTEM_DEFAULT	Die für das System eingestellte Standard-Locale.
LOCALE_USER_DEFAULT	Die für einen bestimmten Anwender eingestellte Standard-Locale (die sich bei Mehrbenutzersystemen von der Systemeinstellung unterscheiden kann).

Siehe auch: Langxxxx Sprach-ID-Konstanten, Typdefinition TLangID

Klasse TOcApp ocf/ocapp.h

Basisklasse

TUnknown

TOcApp ist ein Verbinderobjekt von ObjectComponents für eine Verknüpfungs- und Einbettungsanwendung. Es implementiert die Schnittstellen, die eine Anwendung für die Kommunikation mit OLE benötigt. Jede ObjectComponents-Anwendung, die Verknüpfung und Einbettung unterstützt, muß über ein *TOcApp*-Objekt verfügen. Normalerweise wird es für Sie von Ihrem *TOcRegistrar*-Objekt erzeugt.

Anwendungen, die Automatisierung unterstützen, nicht jedoch Verknüpfung und Einbettung, benötigen kein *TOcApp*-Objekt. Sie erzeugen ein *TRegistrar*-Objekt statt eines *TOcRegistrar*-Objekts.

TOcApp ist ein COM-Objekt und implementiert die Schnittstelle *IUnknown*.

Siehe auch: TOcModule::OnInit, Klasse TRegistrar, Klasse TOleFactory<>, Klasse TOcRegistrar, Klasse TUnknown

Typdefinitionen

TOcMenuEnable

enum TOcMenuEnable

Diese Aufzählungswerte sind Flags, die mit dem bitweisen ODER-Operator (|) kombiniert werden können. Ein Container übergibt sie an die Funktion *EnableEditMenu* um festzustellen, welche OLE-Befehle im Menü Bearbeiten aktiviert werden sollen. Die Antwort hängt davon ab, ob der Container irgendeines der Datenformate unterstützt, die sich aktuell in der Zwischenablage befinden.

Konstante	Aktivierter Menübefehl
EnablePaste	Der Befehl Einfügen plaziert ein Objekt aus der Zwischenablage in dem geöffneten Dokument. Das Format des neuen Datenobjekts wird durch die vom Server bevorzugten und durch die vom Container unterstützten Formate bestimmt.
EnablePasteLink	Der Befehl Verknüpfung einfügen fügt dem geöffneten Dokument eine Verknüpfung auf das Objekt in der Zwischenablage hinzu.
EnableBrowseClipboard	Der Befehl Manuell Einfügen ruft ein Standard-Dialogfenster auf, das alle Datenformate anzeigt, die in der Zwischenablage aktuell für das Objekt zur Verfügung stehen, und läßt den Anwender daraus auswählen.
EanbleBrowseLinks	DerBefehl Verknüpfungen zeigt eine Liste aller verknüpften Objekte im geöffneten Dokument an und erlaubt es dem Benutzer, sie zu aktualisieren oder zu löschen.

Siehe auch: Makro REGFORMAT, TOcApp::EnableEditMenu

Öffentliche Elementfunktionen

AddUserFormatName

void AddUserFormatName(char far* name, char far* resultName, char far* id = 0);

Diese Funktion verbindet einen Ergebnisnamen mit einem Zwischenablageformat. Der Parameter *resultName* beschreibt Anwendern das Datenformat und erscheint im Hilfetext des Dialogfensters Manuell Einfügen. Verwenden Sie einen der beiden anderen Parameter, um das verbundene Zwischenablageformat zu identifizieren. Diese Methode wird nur eingesetzt, wenn Sie ein vom Standard abweichendes, privates Zwischenablageformat verwenden, das Sie mit Namen, die im Dialogfenster Manuell Einfügen verwendet werden, verbinden möchten.

Ein benutzerdefiniertes Format muß zunächst in den Registriertabellen der Anwendung mit dem Makro REGFORMAT eingetragen werden. Beispielsweise

```
REGFORMAT(0, "DrawingClip", ocrContent, ocrIstorage, ocrGet);
```

"DrawingClip" wird der ID-String, den Windows intern verwendet, um das benutzerdefinierte Format zu identifizieren. Um weitere beschreibende Strings mit dem benutzerdefinierten Format zu verknüpfen, rufen Sie *AddUserFormatName* auf:

```
AddUserFormatName("DrawingPad", "Freihandzeichnung", "DrawingClip");
```

Der Name des Formats "DrawingClip" ist nun "DrawingPad". Wenn sich Daten dieses Typs in der Zwischenablage befinden und der Benutzer Manuell Einfügen auswählt, ist der Name im Dialogfenster "DrawingPad". Es ist aber auch zulässig, für ID und Namen die gleiche Zeichenkette zu verwenden.

Der Ergebnis-String, "Freihandzeichnung", erscheint typischerweise im Hilfetext während eines manuellen Einfügevorgangs.

Siehe auch: Makro REGFORMAT

Browse

bool Browse(TOcInitInfo& initInfo);

Zeigt das Dialogfenster Objekt Einfügen an, das dem Benutzer die Auswahl unter den verfügbaren Servern gestattet, wenn ein neues Objekt in dem geöffneten Dokument erzeugt werden soll. Gibt **true** zurück, wenn der Anwender ein Objekt einfügt und **false**, wenn er den Vorgang abbricht.

Erzeugen Sie zunächst *initInfo*, indem Sie seinem Konstruktor das View-Objekt übergeben, in das das neue Objekt eingefügt werden soll. *Browse* übergibt *initInfo* Informationen über das Objekt. Verwenden Sie dann *initInfo*, um ein neues *TOcPart*-Objekt zu erzeugen.

Siehe auch: TOcInitInfo, TOcPart, TOcView

Browse Clipboard

bool BrowseClipboard(TOcInitInfo& initInfo);
Zeigt das Dialogfenster Manuell Einfügen an, in dem die verfügbaren Formate für die sich aktuell in der Zwischenablage befindenden Formate verzeichnet sind, und erlaubt dem Benutzer, das Format für den Einfügevorgang auszuwählen. Gibt **true** zurück, wenn der Anwender Daten einfügt und **false**, wenn er den Vorgang abbricht.

Erzeugen Sie zunächst *initInfo*, indem Sie seinem Konstruktor das View-Objekt übergeben, in das das neue Objekt eingefügt werden soll. *Browse* übergibt *initInfo* Informationen über das Objekt. Verwenden Sie dann *initInfo*, um ein neues *TOcPart*-Objekt zu erzeugen.

Diese Funktion wird von *TOcView::BrowseClipboard* aufgerufen.

Siehe auch: TOcInitInfo, TOcPart, TOcView

CanClose

bool CanClose();
Ein Container ruft diese Funktion auf um festzustellen, ob er schließen kann. *CanClose* fragt alle aktuell verbundenen Server ab und versucht, sie zu schließen. Liefert **true** zurück, wenn die Anwendung sicher geschlossen werden kann.

Clip

bool Clip(IBPart far* part, bool link, bool embed, bool delay = false);
Kopiert das aktuell ausgewählte Objekt in die Zwischenablage. Normalerweise müssen Sie *Clip* nicht direkt aufrufen, weil *TOcView::Copy* das für Sie übernimmt.

part zeigt auf das verknüpfte oder eingebettete Objekt. Sie können für diesen Parameter ein Objekt des Typs *TOcPart* übergeben. (*TOcPart* unterstützt die Schnittstelle *IBPart*, die in der BOCOLE-Bibliothek definiert ist). Wenn sowohl *link* als auch *embed* **true** sind, können andere Anwendungen das Objekt verknüpfen oder es einbetten, wenn sie es aus der Zwischenablage einfügen. Setzen Sie *delay* auf **true**, wenn ObjectComponents alternative Datenformate verzögert zur Verfügung stellen soll. Das verzögerte

Bereitstellen alternativer Datenformate spart Arbeitsspeicher. Näheres dazu finden Sie in der API-Hilfedatei.

Siehe auch: TOcPart, TOcView::Copy

Convert

bool Convert(TOcPart* ocPart, bool activate);
Zeigt das Dialogfenster Konvertieren an, in dem ein Anwender die Darstellung oder das Format eines verknüpften oder eingebetteten Objekts ändern kann. *ocPart* zeigt auf das Objekt, das der Benutzer ändern möchte.

Setzen Sie *activate* auf **true**, wenn ObjectComponents das Objekt nach der Konvertierung aktivieren soll. Im allgemeinen sollte *activate* **false** sein, wenn der Anwender Verknüpfungen aus dem Menü Bearbeiten ausgewählt hat. Wenn der Anwender versucht, ein Objekt zu aktivieren, dessen Server nicht verfügbar ist, können Sie die Möglichkeit anbieten, das Objekt auf einen anderen Server zu konvertieren, und in diesem Fall sollte *activate* **true** sein.

Siehe auch: TOcPart

Drag

bool Drag(IBPart far* part, TOcDropAction inAction, TOcDropAction& outAction);
Ein Container ruft diese Funktion auf, wenn der Benutzer eines der Objekte des Containers mit der Maus ziehen möchte. Der erste Parameter, *part*, ist das Objekt, das der Anwender zu ziehen versucht. Normalerweise ist dies ein Objekt vom Typ *TOcPart*. (*TOcPart* unterstützt die Schnittstelle *IBPart*, die in der BOCOLE-Bibliothek definiert ist).

inAction kombiniert Bitflags, die mögliche Mauszieh-Aktionen anzeigen, die die Anwendung unterstützt. Die Flags zeigen an, ob der Anwender das Objekt bewegen, verknüpfen oder kopieren kann. Der in *outAction* zurückgegebene Wert enthält nur eines der Aktionsflags, das anzeigt, welche Aktion tatsächlich ausgeführt wurde.

Siehe auch: Aufzählungstyp TOcDropAction, TOcPart

EnableEditMenu

uint EnableEditMenu(TOcMenuEnable enable, IBDataConsumer far* ocview);
Eine Anwendung ruft *EnableEditMenu* auf um herauszufinden, welche der auf OLE bezogenen Befehle in ihrem Menü Bearbeiten aktuell aktiviert sein sollten. Die in *enable* kombinierten Flags zeigen die zu überprü-

fenden Befehle an, und der Rückgabewert verwendet die gleichen Bitflags um anzuzeigen, welche Befehle zu aktivieren sind. *ocview* ist normalerweise ein Objekt vom Typ *TOcView* (*TOcView* unterstützt die Schnittstelle *IBDataConsumer*, die in der BOCOLE-Bibliothek definiert ist).

TOleContainer und *TOleView* rufen *TOcApp::EnableEditMenu* in den Befehlsakivierungsfunktionen für das Menü Bearbeiten auf.

Siehe auch: Aufzählungstyp TOcApp::TOcMenuEnable, TOcView

EvActivate

void EvActivate(bool active);
Ein Container ruft diese Funktion auf um OLE zu informieren, wenn sein Rahmenfenster aktiv oder inaktiv wird. Setzen Sie *active* auf **true**, wenn das Fenster aktiviert wurde und auf **false**, wenn es deaktiviert wurde.

Siehe auch: TOcApp::EvResize, TOcApp::EvSetFocus

EvResize

void EvResize();
Ein Container ruft diese Funktion auf um OLE zu informieren, daß sich die Größe seines Rahmenfensters (des Hauptfensters) geändert hat. OLE benötigt diese Information unter Umständen, damit ein Server beim direkten Bearbeiten im Container-Dokument seine Mauspalette ändern kann.

Siehe auch: TOcApp::EvActivate, TOcApp::EvSetFocus

EvSetFocus

bool EvSetFocus(bool set);
Ein Container ruft diese Funktion auf um OLE zu informieren, daß sein Rahmenfenster den Fokus entweder erhalten oder ihn abgegeben hat. Setzen Sie *set* auf **true**, wenn das Fenster den Fokus erhalten und auf **false**, wenn es ihn verloren hat.

Siehe auch: TOcApp::EvActivate, TOcApp::EvResize

GetName

string GetName() const;
Liefert ein String-Objekt zurück, das den Namen der Anwendung enthält.

GetNameList

TOcNameList& GetNameList();
Liefert ein Array von *TOcNameList*-Objekten zurück, das die Namen aller Zwischenablageformate enthält, die die Anwendung unterstützt. Die Klasse *TOcView* verwendet diese Liste bei der Ausführung des Befehls Manuell Einfügen. Die Liste stellt die Namen und Hilfe-Strings zur Verfügung, die mit den Formaten verbunden sind.

Siehe auch: TOcNameList

GetRegistrar()

TOcRegistrar& GetRegistrar();
Liefert das Registrar-Objekt der Anwendung zurück. Dies ist das gleiche Objekt, das in den *TOcApp*-Konstruktor übergeben wird.

Siehe auch: TOcApp öffentlicher Konstruktor und Destruktor, Klasse TOcRegistrar

IsOptionSet

bool IsOptionSet(uint32 option) const;
Überprüft die Anwendungsmodus-Flags und gibt **true** zuück, wenn die in *option* gesetzten Flags auch für die Anwendung gesetzt sind. Die Anwendungsmodus-Flags sind im Aufzählungstyp *TOcAppMode* definiert.

Paste

bool Paste(TOcInitInfo& initInfo);
Übergibt *initInfo* Informationen über das Objekt in der Zwischenablage. Liefert **true** zurück, wenn es Informationen ermittelt und **false**, wenn das mißlingt.

Erzeugen Sie zunächst *initInfo*. Der *TOcInitInfo*-Konstruktor erhält das View-Objekt, in das der neue Teil eingefügt werden soll. Rufen Sie dann *Paste* auf, um Informationen in *initInfo* einzusetzen. Rufen Sie abschließend *TOcView::Drop* auf, um das Objekt in die View einzufügen.

Diese Funktion wird von *TOcView::Paste* aufgerufen.

Siehe auch: TOcInitInfo, TOcView, TOcView::Drop, TOcView::Paste

RegisterClass

bool RegisterClass(const string& progid, BCID classId, bool multiUse);
Informiert OLE, daß die Anwendung Objekte eines bestimmten Typs erzeugen kann. Welche Objekte ein Server erzeugen kann hängt von den Dokumenttypen ab, die er registriert.

progid ist der registrierte String, der einen Objekttyp identifiziert.

RegisterClasses iteriert schleifenförmig über die Dokument-Templates der Anwendung und ruft für jeden Typ einmal *RegisterClass* auf. Der Aufruf geschieht intern, und normalerweise müssen Sie keine der Funktionen direkt aufrufen.

Siehe auch: TOcApp::RegisterClasses, TOcApp::UnregisterClass, TOcApp::UnregisterClasses

RegisterClasses

void RegisterClasses(const TDocTemplate* tplHead = ::DocTemplateStaticHead);
Gibt OLE bekannt, daß die Anwendung ausgeführt wird und informiert OLE über alle von der Anwendung registrierten Dokumenttypen. Die Dokumenttypen werden OLE als Objektarten zugänglich gemacht, die die Anwendung erzeugen kann.

tplHead zeigt auf den Anfang der Liste von Dokument-Templates der Anwendung. ObjectWindows speichert diese Liste in der globalen Variablen *DocTemplateStaticHead*.

UnregisterClasses iteriert schleifenförmig über die Liste der Dokumenttypen und ruft *UnregisterClass* für jeden auf, der über eine registrierte *progid* verfügt.

RegisterClasses iteriert schleifenförmig über die Dokumentstrukturen in *tplHead* und ruft *RegisterClass* einmal für jeden Typ auf, der über eine *progid* verfügt. Der Aufruf geschieht intern, und normalerweise müssen Sie keine der Funktionen direkt aufrufen.

Siehe auch: Registrierschlüssel progid, TOcApp::RegisterClass, TOcApp::UnregisterClass, TOcApp::UnregisterClasses

ReleaseObject

virtual void ReleaseObject();
ReleaseObject teilt dem Objekt mit, daß das Hauptfenster der Anwendung nicht mehr existiert. Wenn die Anwendung nicht einem Client als Server dient, verringert *ReleaseObject* darüber hinaus den internen Referenzzäh-

ler des *TOcApp*-Objekts. Wenn der Zähler 0 erreicht, zerstört das Objekt sich selbst. Dert Destruktor von *TOcModule* ruft diese Funktion auf.

Siehe auch: TOcModule

SetOption

void SetOption(uint32 bit, bool state);
Modifiziert die Ausführungsmodus-Flags der Anwendung. *bit* enthält Bitflags aus dem Aufzählungstyp *TOcAppMode*. Wenn *state* **true** ist, schaltet *SetOption* die Flags ein. Ist **state false**, schaltet es die Flags aus. Es sollte niemals erforderlich werden, diese Funktion direkt aufzurufen, weil ObjectComponents die Modus-Flags selbsttätig pflegt.

Siehe auch: TOcApp::IsOptionSet, Aufzählungstyp TOcAppMode

SetupWindow

void SetupWindow(HWND frameWnd);
Teilt dem *TOcApp*-Objekt mit, welches Fenster es mit der Anwendung verbinden soll. Normalerweise ist *frameWnd* das Hauptfenster der Anwendung. Für gewöhnlich wird diese Funktion von der Funktion *SetupWindow* aufgerufen, die mit dem Hauptfenster der Anwendung verbunden ist.

TranslateAccel

bool TranslateAccel(MSG far* msg);
Eine Container-Anwendung fügt ihrer Windows-Botschaftenschleife *TranslateAccel* hinzu, wenn sie dem Anwender die Tastenkürzel eines DLL-Servers während des direkten Bearbeitens im Container-Dokument zur Verfügung stellen will. DLL-Server sind auf diese Form der Zusammenarbeit angewiesen, weil sie im Gegensatz zu .EXE-Servern keine eigenen Botschaftenschleifen haben.

Wenn Sie *TranslateAccel* nach dem üblichen Aufruf an *TranslateAccelerator* aus der Windows-API aufrufen, haben Ihre eigenen Tastenkürzel im Konfliktfall Vorrang vor denen des Servers.

msg enthält eine Windows-Botschaftenstruktur. Der Rückgabewert ist **true**, wenn der Server das Tastenkürzel übersetzt und **false**, wenn er das nicht tut.

UnregisterClass

bool UnregisterClass(const string& progid);
Benachrichtigt OLE, wenn eine Anwendung nicht länger verfügbar ist, um Objekte eines bestimmten Typs zu erzeugen. *progid* ist der registrierte String, der einen Objekttyp identifiziert.

UnregisterClasses iteriert schleifenförmig durch alle Dokumente, die die Anwendung registriert hat und ruft *UnregisterClass* für jedes einzelne auf. Der Destruktor von *TOcApp* ruft *UnregisterClasses* auf.

Siehe auch: TOcApp::RegisterClass, TOcApp::RegisterClasses, TOcApp::UnregisterClasses

UnregisterClasses

void UnregisterClasses(const TDocTemplate* tplHead = ::DocTemplateStaticHead);
Teilt dem System mit, daß die Anwendung nicht länger für OLE-Transaktionen zur Verfügung steht. *tplHead* zeigt auf den Anfang der Liste von Dokument-Templates der Anwendung. ObjectWindows speichert diese Liste in der globalen Variablen *DocTemplateStaticHead*.

UnregisterClasses iteriert schleifenförmig über die Liste der Dokumenttypen und ruft *UnregisterClass* für jeden auf, der über eine registrierte *progid* verfügt. *UnregisterClasses* wird vom *TOcApp*-Destruktor aufgerufen.

Siehe auch: Registrierschlüssel progid, TOcApp::RegisterClass, TOcApp::RegisterClasses, TOcApp::UnregisterClass

Geschützter Konstruktor und Destruktor

Konstruktor

TOcApp(TOcRegistrar& registrar, uint32 options = ULONG_MAX, IUnknown* outer = 0, const TDocTemplate* tplHead = ::DocTemplateStaticHead);
Der Konstruktor für ein *TOcApp*-Objekt erweitert, falls erforderlich, die Botschaften-Queue der Anwendung, um Platz für den OLE-Botschaftenverkehr zur Verfügung zu stellen, und erzeugt die Liste der von der Anwendung unterstützten Zwischenablageformate.

- registrar ist ein Registrierungsobjekt, das die Befehlszeile verarbeitet. Erzeugen Sie das registrar-Objekt zuerst.

options ist eine Menge von Anwendungsmodus-Bitflags. Das *TOcApp*-Objekt wird normalerweise in der Callback-Funktion *TComponentFactory* erzeugt. Der Parameter *options* des Konstruktors ist der gleiche wie der Parameter *options* der Callback-Funktion.

outer zeigt auf die Schnittstelle *IUnknown* des äußeren Objekts. Die Anwendung ist aufgefordert, sich innerhalb dieses Objekts zu versammeln (Aggregation).

tplHead zeigt auf den Anfang einer Liste von Dokument-Templates der Anwendung. Die ObjectWindows-Bibliothek speichert die Liste der Dokument-Templates einer Anwendung in der globalen Variablen *DocTemplateStaticHead*.

Destruktor

~TOcApp();
Der *TOcApp*-Destruktor teilt OLE mit, daß die Anwendung nicht länger verfügbar ist.

Normalerweise wird die Erzeugung und Entfernung des *TOcApp*-Objekts einer Anwendung vom *TOcRegistrar*-Objekt verwaltet.

Siehe auch: Typdefinition TComponentFactory, Klasse TOcRegistrar, TOcApp::ReleaseObject

Geschützte Elementfunktionen

ForwardEvent

uint32 ForwardEvent(int eventid, const void far* param);
uint ForwardEvent(int eventid, uint32 param = 0);
Beide Formen senden eine WM_OCEVENT-Botschaft an das Hauptfenster der Anwendung. Der Parameter *eventId* wird zum *wParam* der Botschaft. Er sollte eine der Konstanten OC_APPxxxx oder OC_VIEWxxxx sein. Der zweite Parameter wird zum *lParam* der Botschaft. Er kann entweder ein Zeiger (Form 1) oder ein Integerwert (Form 2) sein. Welche Form gewählt wird, hängt davon ab, welche Informationen ein bestimmtes Ereignis benötigt, um seinen *lparam* zu senden.

Siehe auch WM_OCEVENT-Botschaft, OC_APPxxxx-Botschaften, OC_VIEWxxxx-Botschaften, Klasse TOcRegistrar

Aufzählungstyp TOcAspect ocf/ocobject.h

enum TOcAspect
Ein Container verwendet diese Werte, um anzufordern, daß Objekte in seinen Dokumenten auf bestimmte Weise dargestellt werden. Ein Objekt kann angewiesen werden, seinen gesamten Inhalt darzustellen, eine verkleinerte Darstellung seines Inhalts anzuzeigen oder ein Symbol, das sei-

nen Objekttyp repräsentiert. Ein Server muß nicht alle möglichen Ansichten unterstützen.

Die Werte sind Flags und können mit dem bitweisen Operator OR (|) kombiniert werden.

Konstante	Bedeutung
asContent	Zeigt den gesamten Inhalt des Objekts in normaler Größe.
asThumbnail	Zeigt den Inhalt des Objekts so verkleinert, daß er auf eine kleinere Fläche paßt.
asIcon	Zeigt ein Symbol, das den Objekttyp repräsentiert.
asDocPrint	Zeigt das Objekt so wie es aussieht, wenn es an den Drucker geleitet wird.
asDefault	Verwendet weiterhin die zuletzt angegebene Größe.
asMaintain	Erhält das Original-Seitenverhältnis des Objekts. Paßt das Seitenverhältnis nicht an das Rechteck an, in dem der Client das Objekt anzeigt.

Siehe auch TOcPart::Draw, Aspektkonstanten ocrxxxx

Aufzählungstyp TOcAppMode ocf/ocreg.h

enum TOcAppMode

Die Aufzählungswerte von *TOcAppMode* stellen Flags dar, die von ObjectComponents gesetzt werden, um die Ausführungsmodi einer Anwendung anzugeben. Einige Flags werden als Antwort auf Kommandozeilen-Schalter gesetzt, die OLE auf der Kommandozeile eines Servers ausführt; andere werden gesetzt, während sich die Anwendung registriert.

Man kann durch den Aufruf von *TOcApp::IsOptionSet* oder *TOcModule::IsOptionSet* feststellen, ob ein bestimmtes Flag gesetzt ist. Das Objekt *TOcApp* enthält die Modus-Flags für jede Instanz der Anwendung. *TOcModule* fragt einfach *TOcApp* ab.

Die Aufzählungswerte sind Bitflags und können mit dem bitweisen Operator OR (|) kombiniert werden. Durch einen Stern gekennzeichnete Flags können sich für jede Instanz einer Anwendung unterscheiden.

Konstante	Wie der Server reagiert
amAnyRegOption	Kombiniert die Bits aus *RegServer*, *UnregServer* und *TypeLib*.
amAutomation	Server registriert sich als Einzelbenutzer (nur ein Client). Wird immer mit -Embedding verwendet.
amDebug	Startet eine Debug-Sitzung.
amExeMode	*Ohne Reaktion. Dieses Flag wird gesetzt, um anzuzeigen, daß der Server als .EXE läuft. Entweder wurde der Server als .EXE konstruiert, oder es handelt sich um eine DLL, die von einem .EXE-Stub gestartet wurde und als ausführbares Programm läuft.
amExeModule	Ohne Reaktion. Dieses Flag wird gesetzt, um anzuzeigen, daß der Server als .EXE-Programm konstruiert wurde.
amEmbedding	*Bleibt verborgen, da es für einen Client und nicht für sich selbst ausgeführt wird.

amLangId	Server verwendet bei der Erstellung von Registrier- und Typbibliotheken die Locale-ID, die diesem Schalter folgt. (Ohne den -RegServer- oder -TypeLib-Schalter nutzlos.)
amNoRegValidate	Läßt die übliche Gültigkeitsprüfung aus, die *progid*, *clsid* und *path* des Servers mit den Registrierungen des Systems vergleicht. Das Registrar-Objekt antwortet auf dieses Flag.
amRegServer	Server registriert sich in der Registrierungsdatenbank des Systems und beendet sich.
amRun	Server führt seine Meldungsschleife aus. Wird von der Factory-Callback-Funktion verwendet.
amServedApp	*Verhindert, daß der Server sich selbst löscht (ein Client benutzt die Anwendung und besitzt eine Referenz zu ihr).
amShutdown	*Entdeckt der Callback *TComponentFactory* dieses Flag, sollte er die Anwendung beenden.
amSingleUse	*Server registriert sich als Einzelbenutzer-Anwendung (nur ein Client).
amTypeLib	Erzeugen und Registrieren einer Typbibliothek.
amUnregServer	Server entfernt alle Einträge aus der Registrierungsdatenbank des Systems und beendet sich.

Siehe auch TOcModule::IsOptionSet, TOcApp::IsOptionSet

Aufzählungstyp TOcDialogHelp ocf/ocobject.h

enum TOcDialogHelp

Das Ereignis OC_APPDIALOGHELP teilt dem Container mit, daß ein Anwender auf den Hife-Schalter in einem OLE-Standarddialogfeld klickt. Der *Iparam* der Botschaft WM_OCEVENT enthält einen dieser Werte und zeigt damit an, welches Dialogfeld der Anwender geöffnet hat.

Konstante	Dialogfeld	Funktion
dhBrowse	Dialogfeld Objekt einfügen	Wählt einzufügendes Objekt aus
dhBrowseClipboard	Dialogfeld Inhalte einfügen	Wählt das Datenformat für das Einfügen eines Objekts aus.
dhConvert	Dialogfeld Konvertieren	Konvertiert ein Objekt, damit es mit einem anderen Server verwendet werden kann.
dhBrowseLinks	Dialogfeld *Verknüpfungen*	Verknüpfungen zu Objekten aktualisieren.
dhChangeIcon	Dialogfeld *Symbol ändern*	Wird intern von den Dialogfeldern Objekt einfügen und Inhalte einfügen verwendet.
dhFileOpen	Dialogfeld *Datei öffnen*	Wählt eine Datei zum Öffnen aus.
dhSourceSet	Dialogfeld *Quelle ändern*	Weist einem verknüpften Objekt eine neue Verknüpfungsquelle zu.
dhIconFileOpen	Dialogfeld *Datei Öffnen*	Bestätigt, daß die ausgewählte Datei eine Symbol-Ressource enthält.

Siehe auch Ereignisbehandlungsroutine EvOcAppDialogHelp, OC_APxxxx-Botschaften, TOleFrame::EvOcAppDialogHelp, WM_OCEVENT-Botschaft

Klasse TOcDocument ocf/ocdoc.h

Die Hauptaufgabe von *TOcDocument* ist es, Daten unter Verwendung hierarchisch geordneter Speicherabteile in einer Verbunddatei zu speichern und zu laden. (Ein Speicherabteil ist ein Abteil innerhalb einer Datei - so wie ein Verzeichnis ein Abteil auf einer Festplatte darstellt.)

Standardmäßig werden die nativen Daten der Anwendung immer im Stammspeicherabteil des Dokuments abgelegt. Die Anwendung kann jedoch auch ihre eigenen Speicherabteile in der gleichen Datei anlegen. *TOcDocument* erzeugt bei Bedarf neue Speicherabteile unterhalb des Stammspeicherabteils für OLE-Objekte, die der Anwender in das Verbunddokument einfügt. Die neuen Speicherabteile erhalten ihre Namen von den Namen der Objekte, die sie speichern. *TOcView* weist automatisch jedem neuen Objekt einen eindeutigen String-Bezeichner zu.

Sowohl Server wie auch Container können Objekte vom Typ *TOcDocument* erzeugen. Im Container stellt dieses Objekt ein vollständiges Verbunddokument dar. Im Server stellt es die Daten für ein einzelnes OLE-Objekt dar. (Andere OLE-Objekte können mit dem einzelnen OLE-Objekt des Servers verknüpft oder in es eingebettet sein.)

Ein *TOcDocument*-Objekt verwaltet die Kollektion von *TOcPart*-Objekten, die in einem der Verbunddokumente abgelegt sind. Es zeichnet die Daten nicht auf den Bildschirm. Zu diesem Zweck benötigt jedes *TOcDocument* ein entsprechendes *TOcView*- oder *TOcRemView*-Objekt. Eine Anwendung kann mehrere Paare zugehörige Dokument- und View-Objekte besitzen, eins für jedes geöffnete Dokument.

Ein Container erzeugt ein *TOcView*-Objekt,um sein Verbunddokument im eigenen Fenster des Containers zu zeichnen. Da das Fenster, in dem der Server zeichnet, zum Container gehört (es ist ein untergeordnetes Fenster des Container-Fensters), muß der Server für jedes Dokument ein entferntes View-Objekt (*TOcRemView*) erzeugen.

Obwohl ihre Namen ähnlich sind, gehören *TOcDocument* und *TOcView* nicht zum Dokument-/View-Modell von ObjectWindows. Der Aufbau von OLE macht es sinnvoll, Daten und ihre grafische Darstellung zu trennen. Die Begriffe *Dokument* und *View* drücken diese Trennung sogar außerhalb von ObjectWindows aus.

Ein *TOcDocument* muß die Standard-OLE-Schnittstellen *Istorage* und *Istream* verwenden, um seine Aufgaben auszuführen. Normalerweise müssen diese Schnittstellen nicht direkt angesprochen werden, da ObjectComponents sie in seinen undokumentierten Klassen *TOcStorage* und *TOcStream* implementiert. Diese Klassen stellen Rahmenfunktionen um Standard-OLE-Schnittstellen dar. Die Implementierung von *TOcDocument* verwendet beide Objekte.

Siehe auch

Klasse TOcRemView, Klasse TOcView

Öffentliche Konstruktoren und Destruktor

Konstruktoren

Form 1

TOcDocument(TOcApp& app, const char far* fileName = 0);
Erzeugt ein neues Dokumentobjekt für die Anwendung und weist optional einen Dateinamen für das Speichern des Dokuments zu. Ein Container verwendet diesen Konstruktor für jedes Dokument, das der Anwender öffnet.

Form 2

TOcDocument(TOcApp& app, const char far* fileName, Istorage far* storagel);
Erzeugt ein neues Dokumentobjekt für die Anwendung und weist ein bestimmtes Datei- und Speicherobjekt, das das Dokument enthält, zu. Der Container ruft diesen Konstruktor auf, wenn er eine existierende Datei öffnet. Der Server und der Container erzeugen jeder ihr eigenes *TOcDocument*-Objekt für das Objekt, das sie gemeinsam benutzen. Beide Objekte zeigen jedoch zum Speichern des Objekts auf die gleiche Datei.

IStorage ist die Standard-OLE-Speicherabteilschnittstelle. ObjectComponents implementiert diese Schnittstelle in seiner internen, undokumentierten Klasse *TOcStorage*. Normalerweise muß die Schnittstelle *IStorage* oder die Klasse *TOcStorage* in einer ObjectComponents-Anwendung nicht direkt angesprochen werden.

Destruktor

~TOcDocument();
Zerstört das Objekt *TDocument*.

Öffentliche Elementfunktionen

Close

void Close();
Ein Container ruft *TOcPart::Close* für jedes Objekt im Verbunddokument auf, um seine Server freizugeben. *TOleDocument* ruft diese Funktion bei der Aufforderung zum Schließen automatisch auf.

Siehe auch TOcPart::Close,Klasse TOleDocument

GetActiveView

TOcView* GetActiveView();
Gibt einen Zeiger an die aktive View zurück. *TOcPart* ruft diese Methode auf, um den Fokuswechsel zwischen aktiven Teilen zu koordinieren.

Siehe auch TOcDocument::SetActiveView

GetName

string getName() const;
Gibt den Namen der Datei zurück, in der das Dokument gespeichert wird. ObjectComponents verwendet den Namen, um Verknüpfungen korrekt zu erzeugen.

Siehe auch TOcDocument::SetName

GetParts

TOcPartCollection& GetParts();
Gibt ein Objekt mit Informationen zu allen Teilen des Dokuments zurück. Jeder Teil entspricht einem eingebetteten oder verknüpften Objekt. Erzeugen Sie einen Iterator vom Typ *TOcPartCollectionIter*, um schleifenförmig durch die Kollektion zu iterieren und Information zu bestimmten Teilen zu extrahieren.

Siehe auch Klasse TOcPart, Klasse TOcPartCollection, Klasse TOcPartCollectionIter

GetStorage

TOcStorage* GetStorage();
Gibt das Stammspeicherabteil der Dokumentdatei zurück.

Siehe auch TOcDocument::SetStorage

LoadParts

bool LoadParts();
Liest alle verknüpften und eingebetteten Teile, die in einer Verbunddatei gespeichert sind. *LoadParts* lädt nicht unbedingt sofort alle Daten aus allen Teilen in den Speicher. Die Daten werden nur dann benötigt, wenn das Objekt sichtbar ist.

LoadParts gibt **true** zurück, wenn alle Teile erfolgreich gelesen wurden. Wenn dem Dokument noch keine Datei zugewiesen wurde, wird nichts geladen, und die Funktion gibt noch immer **true** zurück. (Ein Dokument kann eine Datei von seinem Konstruktor, von *SaveToFile* oder von *SetStorage* annehmen.)

Siehe auch Öffentliche Konstruktoren und Destruktoren TOcDocument, TOcDocument::SaveParts, TOcDocument::SaveToFile, TOcDocument::SetStorage

RenameParts

void RenameParts(IBRootLinkable far* BLDocument);
Sollte bei jeder Namensänderung der Dokumentdatei aufgerufen werden. *RenameParts* aktualisiert den internen Namen, der mit jedem Teil gespeichert wird, damit andere Anwendungen sich weiterhin korrekt mit ihnen verknüpfen können.

IBRootLinkable ist eine benutzerdefinierte OLE-Schnittstelle, die in der Support-Bibliothek BOCOLE definiert ist. Objekte vom Typ *TOcView* implementieren diese Schnittstelle. Sie müssen sie also normalerweise nicht selbst implementieren. Übergeben Sie einfach das View-Objekt des Dokuments an *RenameParts*.

TOcView ruft diese Funktion automatisch auf, wenn die View umbenannt wird.

Siehe auch TOcDocument::SetName, TOcView::Rename

SaveParts

bool SaveParts(IStorage* storage = 0, bool sameAsLoaded = true);
Schreibt alle verknüpften und eingebetteten Objekte des Dokuments in die Dokumentdatei. *storage* ist das Stammspeicherabteil der Datei. Das *TOcDocument* eines Containers erzeugt das Speicherateil-Objekt beim Erzeugen oder ersten Speichern des Dokuments. Das Objekt kann mit *GetStorage* gesucht werden. Ein Server erhält das Speicherabteil-Objekt vom Container. Normalerweise ist es nicht nötig, das Objekt *storage* direkt anzusprechen.

Solange der Dokumentname seit dem letzten Laden oder Speichern des Dokuments nicht geändert wurde, sollte *sameAsLoaded* **true** sein.

- *SaveParts* gibt **true** zurück, wenn alle Objekte erfolgreich in die Datei geschrieben wurden.
- *LoadParts* und *SaveParts* werden durch die Methoden *Open* und *Commit* in *TOleDocument* aufgerufen.

Siehe auch TOcDocument::GetStorage, TOcDocument::LoadParts, TOcDocument::SaveToFile

SaveToFile

bool SaveToFile(const char far* newName);
Speichert das Dokument in der Datei, die durch *newName* spezifiziert wird. Normalerweise ruft ein Container diese Funktion auf, wenn der Anwender Datei | Speichern für ein unbenanntes Dokument oder Datei |

Speichern Unter für beliebige Dokumente wählt. *SaveToFile* erzeugt ein neues Speicherabteil-Objekt und ruft dann *SaveParts* auf. Der Rückgabewert ist **true**, wenn alle verknüpften und eingebetteten Teile erfolgreich gespeichert wurden.

Siehe auch TOcDocument::SaveParts

SetActiveView

void SetActiveView(TOcView* view);
Ein *TOcView*-Objekt ruft diese Methode auf, wenn es aktiviert wird. Das Dokument kann dann die aktive View feststellen. Die aktive View schickt Botschaften an das zugehörige Fenster, möglicherweise ist dies ein *TOleView*-Fenster. Dieses Fenster ist dafür verantwortlich, andere Fenster über Veränderungen zu informieren.

Siehe auch TOcDocument::GetActiveView

SetName

void SetName(const string& newName);
Nennt dem Dokument den Dateinamen, unter dem es gespeichert wird. ObjectComponents muß diesen Namen wissen, um Verknüpfungen korrekt herstellen zu können. Genauer gesagt, veranlaßt *Setname* ObjectComponents, den OLE-Merker, den ein Verknüpfungs-Server zur Verfügung stellen muß, zu aktualisieren.

Siehe auch TOcDocument::GetName

SetStorage

Form 1 **void SetStorage(const char far* path);**
Erzeugt unter Verwendung des Namens in *path* eine Verbunddatei und weist das Stammspeicherabteil der neuen Datei als Stammspeicherabteil des Dokuments zu. Normalerweise ruft ein Container diese Funktion auf, wenn der Anwender Datei | Speichern für ein unbenanntes Dokument oder Datei | Speichern als für beliebige Dokumente wählt.

Form 2 **void SetStorage(IStorage* storage);**
Weist *storage* dem Stammspeicherabteil des Dokuments zu. Normalerweise ruft ein Server diese Funktion auf, wenn ihm vom Container ein *IStorage*-Objekt übergeben wird. (Ein *IStorage*-Objekt implementiert die Standard-OLE-Schnittstelle *IStorage*. Normalerweise ist es nicht erforderlich, auf dieses Objekt direkt zuzugreifen.

Weist dem Dokument eine Speicherabteil zum Schreiben seiner Daten zu. *storage* wird der Stammspeicherabteil des Dokuments. Jedes verknüpfte

oder eingebettete Objekt erhält sein eigenes Speicherunterabteil unter dem Stammspeicherabteil.

Siehe auch TOcDocument::GetStorage

Struktur TOcDragDrop ocf/ocview.h

Enthält Informationen, die eine View oder ein Fenster benötigt, um ein Objekt, das mittels ziehen und ablegen bewegt wird, zu akzeptieren. Die OC_VIEWDRAG- und OC_VIEWDROP-Botschaften enthalten in ihren *IParam*s einen Bezug auf diese Struktur. *TOleView* und *TOleWindow* verarbeiten diese Botschaften. Es sollte also nicht erforderlich sein, *TOcDragDrop* direkt anzusprechen - es sei denn, man programmiert ohne ObjectWindows. Beispiele für die Verarbeitung von OC_VIEWDRAG- und OC_VIEWDROP-Botschaften finden sich im Quellcode der Methoden *EvOcViewDrag* und *EvOcViewDrop* in *TOleView* und *TOleWindow*.

Siehe auch OC_VIEWxxxx-Botschaften, TOleView::EvOcViewDrag, TOleView::EvOcViewDrop, TOleWindow::EvOcViewDrag, TOleWindow::EvOcViewDrop

Öffentliche Elementfunktionen

InitInfo

TOcInitInfo far* InitInfo;
Wenn dieses Feld in einer OC_VIEWDROP-Botschaft enthalten ist, beschreibt es ein Objekt, das gerade auf der View abgelegt werden soll. Das Feld ist Null, wenn es in einer OC_VIEWDRAG-Botschaft enthalten ist.

Siehe auch OC_VIEWxxxx-Botschaften, Klasse TOcInitInfo

Pos

TRect Pos;
Die Koordinaten in *Pos* geben den Bereich der View an, in dem der Anwender ein Objekt abgelegt hat. Die Position wird in Gerätekoordinaten relativ zum Client-Bereich angegeben.

Siehe auch Klasse TRect

Where

TPoint Where;
Die Koordinaten in *Where* geben den Punkt in der View an, an dem die Maus das Objekt freigegeben hat. Die Position wird in Client-Bereichskoordinaten angegeben.

Siehe auch Klasse TPoint

Aufzählungstyp TOcDropAction ocf/ocobject.h

enum TOcDropAction

TOcApp::Drag verwendet diese Werte, um zu beschreiben, welche Aktionen zulässig sind und welche Aktionen tatsächlich während einer Ziehen-und-Ablegen-Aktion auftreten. Die Werte sind Flags und können mit dem bitweisen Operator OR (|) kombiniert werden.

Konstante	Bedeutung
daDropCopy	Objekt an Ablagestelle kopieren.
daDropMove	Objekt an Ablagestelle bewegen.
daDropLink	Verknüpfung mit dem Objekt an der Ablagestelle erzeugen.
daDropNone	Keine Aktion aufgetreten.

Siehe auch TOcApp::Drag

Klasse TOcFormatList ocf/ocview.h

Verwaltet eine Liste der Formate für die Zwischenablage, die eine bestimmte View unterstützt.

TOcFormat, *TOcFormatList* und *TOcFormatListIter* arbeiten zusammen, um die Formatliste zu verwalten. *TOcFormatList* fügt der Liste *TOcFormat*-Objekte hinzu oder löscht sie aus ihr. *TOcFormatListIter* zählt immer dann die Einträge der Liste auf, wenn die View sie nacheinander untersuchen muß. Da *TOcView* die Liste intern erzeugt und verwaltet, muß normalerweise keine dieser Klassen direkt angesprochen werden.

Wenn ObjectComponents die Tabelle mit den Dokumenteinträgen erhält, werden damit Einträge für jedes Zwischenablageformat, das das Dokument erhält oder erzeugt, zur Verfügung gestellt. Aus diesen Einträgen erzeugt *TOcView* eine Objektliste vom Typ *TOcFormat*, in der jedes Objekt ein Format darstellt. Die View benötigt diese Liste, um festzustellen, wann ein Zwischenablage-Befehl oder eine Ziehen-und-Ablegen-Operation erfolgreich sein kann. Wenn beispielsweise der Anwender eine Bitmap auf eine View zieht, die nur Text enthalten kann, weiß *TOcView*, daß das Objekt nicht eingefügt werden kann und paßt den Cursor entsprechend an.

Siehe auch Klasse TOcFormat, Klasse TOcFormatListIter, Klasse TOcView

Öffentlicher Konstruktor und Destruktor

Konstruktor

TOcFormatList();
Erzeugt eine leeres Listenobjekt. Um Einträge in die Liste einzufügen, wird die Methode *Add* aufgerufen.

Destruktor

~TOcFormatList();
Löscht alle Einträge der Liste.

Siehe auch TOcFormatList::Add

Öffentliche Elementfunktionen

Add

int Add(TOcFormat* format);
Fügt der Liste einen neuen Eintrag zum Zwischenablageformat hinzu. Gibt bei Mißlingen 0 und bei Erfolg 1 zurück.

Siehe auch TOcFormatList::Clear

Clear

void Clear(int del = 1);
Entfernt alle Einträge aus der Liste. Wenn *del* 1 ist, löscht *Clear* auch alle *TOcFormat*-Objekte.

Siehe auch TOcFormatList::Add, TOcFormatList::Detach

Count

virtual uint Count() const;
Gibt die Anzahl der Listeneinträge zurück.

Siehe auch TOcFormatList::IsEmpty

Detach

int Detach(const TOcFormat* format, int del = 0);
Entfernt einen Formateintrag aus der Liste. Wenn *del* 1 ist, löscht *Detach* auch das *TOcFormat*-Objekt.

Siehe auch TOcFormatList::Add, TOcFormatList::Clear

Find

unsigned Find(const TOcFormat* format) const;
Durchsucht die Liste nach dem als Format übergebenen Objekt. Wird das Objekt gefunden, gibt *Find* die Position des Objekts in der Liste zurück. (Die erste Position ist 0.) Gibt es *format* in der Liste nicht, gibt *Find* UINT_MAX zurück.

IsEmpty

int IsEmpty() const;
Gibt 1 zurück, wenn das Listenobjekt zur Abfragezeit keine *TOcFormat*-Einträge enthält und 0, wenn die Liste nicht leer ist.

Siehe auch TOcFormatList::Count

operator

TOcFormat*& operator [](unsigned index);
Ermittelt ein Zwischenablageformat durch seine Position in der Liste. Wenn beispielsweise *index* 1 ist, gibt [] den zweiten Eintrag der Liste zurück. Die Reihenfolge der Einträge hängt von der Priorität ab, die ihnen bei der Registrierung zugewiesen wurde.

Klasse TOcFormatListIter ocf/ocview.h

Zählt alle Zwischenablageformate auf, die eine bestimmte View unterstützt.

- *TOcFormat*, *TOcFormatList* und *TOcFormatListIter* arbeiten zusammen, um die Formatliste zu verwalten. *TOcFormatList* fügt der Liste *TOcFormat*-Objekte hinzu oder löscht sie aus ihr.
- *TOcFormatListIter* zählt immer dann die Einträge der Liste auf, wenn die View sie nacheinander untersuchen muß. Da *TOcView* die Liste intern erzeugt und verwaltet, muß normalerweise keine dieser Klassen direkt angesprochen werden.

Wenn ObjectComponents die Tabelle mit den Dokumenteinträgen erhält, werden damit Einträge für jedes Zwischenablageformat, das das Dokument erhält oder erzeugt, zur Verfügung gestellt. Aus diesen Einträgen erzeugt *TOcView* eine Objektliste vom Typ *TOcFormat*, in der jedes Objekt ein Format darstellt. Die View benötigt diese Liste, um festzustellen, wann ein Zwischenablage-Befehl oder eine Ziehen-und-Ablegen-Operation erfolgreich sein kann. Wenn beispielsweise der Anwender eine Bitmap auf eine View zieht, die nur Text enthalten kann, weiß *TOcView*, daß das

Objekt nicht eingefügt werden kann und paßt den Cursor entsprechend an.

Siehe auch

Klasse TOcFormat, Klasse TOcFormatList, Klasse TOcView

Öffentlicher Konstruktor

Konstruktor

TOcFormatListIter(const TOcFormatList& collection)
Konstruiert einen Iterator, um die Zwischenablageformate, die in collection enthalten sind, aufzuzählen.

Öffentliche Elementfunktionen

Current

TOcFormat* Current() const;
Gibt das Format zurück, auf das der Iterator gerade zeigt.

operator++

Form 1

TOcFormat* operator++();
Gibt das aktuelle Format zurück und setzt dann den Iterator herauf, so daß er auf das nächste Format zeigt (Postinkrement).

Form 2

TOcFormat* operator++(int);
Setzt den Iterator herauf, so daß er auf das nächste Format in der Liste zeigt und dieses Format dann zurückgibt (Preincrement).

operator int

operator int() const;
Konvertiert den Iterator in einen Integerwert, um zu überprüfen, ob der Iterator die Aufzählung der Kollektion beendet hat. Wurden Teile nicht aufgezählt, gibt der Operator die aktuelle Position des Iterators in der Teileliste zurück. Hat der Iterator das Listenende erreicht, gibt der Operator Null zurück.

Restart

Form 1

void Restart();
Setzt den Iterator zurück, um nochmals mit dem ersten Format in der Liste zu beginnen.

Form 2

void Restart(unsigned start, unsigned stop);
Setzt den Iterator zurück, um eine Teilmenge der Formatliste aufzuzählen. Es wird mit dem Objekt an der Position start begonnen und bis zu dem Objekt an der Position stop iteriert.

Klasse TOcFormatName ocf/ocapp.h

TOcApp verwendet diese Klasse intern. Sie enthält die Strings, die ein Zwischenablage-Datenformat wie z.B. Text oder Bitmap beschreiben. *TOcApp* zeigt diese Strings in Standard-OLE-Dialogfeldern wie beispielsweise Verknüpfung einfügen an.

Jedes Zwischenablageformat besitzt drei zugehörige Informationsteile: einen ID-Wert, einen Namensstring und einen Ergebnisnamen. Bei Standardformaten ist die ID eine Konstante wie beispielsweise CF_SYLK. Der Namensstring ist ein kurzer Name wie beispielsweise "Sylk". Der Ergebnisname ist ein längerer String, der den Anwender über das Ergebnis des Einfügens der Daten informiert - beispielsweise "ein Datenblatt". Ein *TOcFormatName*-Objekt enthält alle drei Werte für ein Format.

TOcApp erzeugt ein *TOcNameList*-Objekt, das alle benötigten Formatnamen enthält. Es lädt deskriptive Strings in *TOcFormatName*-Objekte und fügt die Objekte eins nach dem anderen seiner Namensliste hinzu. Beide Objekte werden innerhalb von *TOvApp* erzeugt und verwaltet. Normalerweise muß auf diese Objekte nicht direkt zugegriffen werden.

Siehe auch

Klasse TOcApp, Klasse TOcNameList

Öffentliche Konstruktoren und Destruktor

Konstruktoren

Form 1

```
TOcFormatName();
```

Konstruiert ein leeres Formatnamenobjekt.

Form 2

```
TOcFormatName(char far* fmtName, char far* fmtResultName, char far* id = 0);
```

Konstruiert ein Formatnamenobjekt und initialisiert es mit drei Werten, die ein Zwischenablageformat beschreiben. *fmtName* ist der Name des Formats ("ein Windows-Metadatei-Bild"). *id* ist der Wert, den Windows zur Identifizierung des Formats (CF_METAFILEPICT) zuweist. Er wird als Dezimalstring ("3") ausgedrückt.

Destruktor

```
~TOcFormatName();
```

Gibt das Objekt frei.

Öffentliche Elementfunktionen

GetId

```
const char far* GetId();
```

Gibt einen Zeiger auf den String zurück, den das System zur Bestimmung des Formats verwendet.

GetName

```
const char far* GetName();
```

Gibt einen Zeiger auf den Namen des Formats zurück.

GetResult

```
const char far* GetResultName();
```

Gibt den deskriptiven String zurück, der den Benutzer über das Ergebnis des Einfügens von Daten dieses Formats informiert.

operator =

```
bool operator==(const TOcFormatName& other);
```

Gibt **true** zurück, wenn *other* das gleiche Objekt wie **this** ist.

Aufzählungstyp TOcInitHow — ocf/ocobject.h

enum TOcInitHow
Diese Werte geben Auskunft darüber, ob ein Container die Objekte, die er empfängt, verknüpfen oder einbetten soll. Der Container übergibt diese Information an ein *TOcInitInfo*-Objekt, wenn er ein neues OLE-Objekt erhält.

Konstante	Bedeutung
ihLink	Verknüpfung mit dem Objekt. Erzeugt im Container-Dokument eine Referenz, die auf die Stelle des Server-Dokuments zeigt, an der sich die Daten tatsächlich befinden.
ihEmbed	Objekt einbetten. Kopiert die Daten des Objekts direkt in das Container-Dokument.
ihMetafile	Einbetten eines statischen Objekts, das sich selbst als Metadatei zeichnet.
ihBitmap	Einbetten eines statischen Objekts, das sich selbst als Bitmap zeichnet.

Siehe auch Öffentliche Konstruktoren TOcInitInfo, Aufzählungstyp TOcInitWhere

Klasse TOcInitInfo — ocf/ocobject.h

TOcInitInfo enthält Informationen, die ObjectComponents darüber informieren, wie ein neuer Teil zu erstellen ist. Wenn der Anwender ein Objekt

in einen Container kopiert, einfügt oder es in ihm ablegt, erzeugt ObjectComponents ein *TOcInitInfo*-Objekt, initialisiert dieses mit den Informationen zum eingehenden OLE-Objekt und übergibt das Info-Objekt an den Konstruktor von *TOcPart*. Das Info-Objekt meldet dem Teil, wo seine Daten zu finden sind und wie er sich selbst erzeugen kann.

TOleView verwaltet diese Einzelheiten, wenn man mit ObjectWindows arbeitet. Wenn Sie ohne ObjectWindows programmieren, finden Sie Beispielcode für die Verwendung von *TOcInfo*-Objekten in den *TOleView*-Methoden, die Objekte einfügen: Sehen Sie sich den Code für *CmEditInsertObject* und *CmEditPasteSpecial* an, wie auch den Code für *TOcView::Drop*.

Siehe auch

Klasse TOcPart, TOcView::Drop, TOleView::CmEditInsertObject, TOleView::CmEditPasteSpecial

Öffentliche Datenelemente

Container

IBContainer far* Container;

Container ist das View-Objekt, das das Objekt empfangen soll. *IBContainer* ist eine undokumentierte, benutzerspezifische OLE-Schnittstelle, die in der Support-Bibliothek BOCOLE definiert und in *TOcView* implementiert ist. Das Datenelement *Container* kann ein Objekt des Typs *TOcView* enthalten.

Siehe auch

Klasse TOcView

HIcon

HICON HIcon;

HIcon enthält das Symbol zum Zeichnen, wenn der Anwender aus dem Dialogfeld *Objekt einfügen* die Option *Als Symbol anzeigen* auswählt. Das Handle von *HIcon* ist ein globales Speicher-Handle zu einer Metadatei, die das Symbol enthält. Die Funktionen *Browse* und *BrowseClipboard* in *TOcApp* verwalten das Dialogfeld Objekt einfügen. Sie müssen also normalerweise das Symbol nicht selbst darstellen.

How

TOcInitHow How;

Gibt an, ob das Objekt beim Hinzufügen zum Dokument verknüpft oder eingebettet werden soll.

Siehe auch

Aufzählungstyp TOcInitHow

Storage

IStorage far* Storage;
Storage ist das Speicherabteil-Objekt in einer Verbunddatei. Der Container stellt das Speicherabteil für die vom Server übertragenen Daten zur Verfügung. *IStorage* ist eine Standard-OLE-Schnittstelle.

ObjectComponents implementiert die Schnittstelle *Istorage* in *TOcStorage*. Daher enthält *Storage* normalerweise ein *TOcStorage*-Objekt.

Where

TOcInitWhere Where;
Gibt an, wo der Server die Objektdaten ablegen wird. Beispielsweise kann der Server die Daten durch Ablage in einer Datei, einem Speicherabteil oder einem Speicher-Handle übertragen.

Siehe auch

Aufzählungstyp TOcInitWhere

Data

IDataObject* Data;
Eines von vier Datenfeldern in einem unbenannten Verbund. Diese Feld wird dann benutzt, wenn *Where iwDataObject* ist. Es besagt, daß der Server ein OLE-Datenobjekt erzeugt hat, um die Daten des eigehenden Objekts zu übertragen. *Data* zeigt auf die Schnittstelle *IDataObject* auf dem Datenübertragungsobjekt des Servers. (*IDataObject* ist eine Standard-OLE-Schnittstelle.) Dies ist die übliche Übertragungsmethode für Objekte, die von der Zwischenablage oder durch eine Ziehen-und-Ablegen-Operation empfangen wurden.

Path

LPCOLESTR Path;
Eines von vier Datenfeldern in einem unbenannten Verbund. Diese Feld wird dann benutzt, wenn *Where iwFile* ist. Es besagt, daß der Server die Daten für das eingehende Objekt in einer Datei abgelegt hat. *Path* zeigt auf den Namen der Datei, in der die Daten gespeichert sind.

Cld

BCID Cld;
Eines von vier Datenfeldern in einem unbenannten Verbund. Dieses Feld wird dann benutzt, wenn *Where iwNew* ist. Es besagt, daß das eingehende Objekt ganz neu ist und gerade erzeugt wurde. *Cld* ist die Klassen-ID, die

der Server für eine seiner Dokumentquellen registriert hat. Sie informiert den Server darüber, welche Art von Objekt zu erzeugen ist.

Siehe auch

TOcApp::RegisterClasses, TOcInitInfo::Where

Handle

```
struct{
  HANDLE Data;
  unint DataFormat;
} Handle;
```

Eines von vier Datenfeldern in einem unbenannten Verbund. Diese Feld wird dann benutzt, wenn *Where iwHandle* ist. Es besagt, daß der Server die Daten für das eingehende Objekt in einem Speicher-Handle abgelegt hat. *Data* ist das Handle selbst, und *DataFormat* kennzeichnet ein Zwischenablageformat für die Daten im Handle.

Öffentliche Konstruktoren

TOcInitInfo

IBContainer far* Container;

Container ist das View-Objekt, das das Objekt empfangen soll. *IBContainer* ist eine undokumentierte, benutzerspezifische OLE-Schnittstelle, die in der Support-Bibliothek BOCOLE definiert und in *TOcView* implementiert ist. Das Datenelement *Container* kann ein Objekt des Typs *TOcView* enthalten.

Siehe auch

Klasse TOcView

TOcInitInfo

Form 1

TOcInitInfo(IBContainer far* container);

Form 1 sollte verwendet werden, wenn der Server aufgerufen wird, um ein ganz neues Objekt zu erzeugen - beispielsweise bei der Verarbeitung des Befehls Objekt einfügen. Der neue Teil wird eingebettet, nicht verknüpft.

Form 2

```
TOcInitInfo(TOcInitHow how, TOcInitWhere where, IBContainer far*
container);
```

Form 2 sollte verwendet werden, wenn ein Teil erzeugt wird, der ein bereits bestehendes Objekt enthält - beispielsweise beim Laden eines Teils aus dem Speicher in ein Verbunddokument. *how* gibt an, ob das Objekt verknüft oder eigebettet wird. *where* gibt an, welches Medium der Server verwenden wird, um die Daten des bestehenden Objekts zu übertragen.

Beide Formen des Konstruktors ereugen ein *TOcInitInfo*-Objekt zum Ablegen eines neuen Teils in *container*. *container* ist die View, die den neuen

Teil enthalten wird. *IBContainer*ist eine benutzerspezifische OLE-Schnittstelle, die in der Support-Bibliothek BOCOLE definiert und in *TOcView* implementiert ist. *container* kann ein Objekt des Typs *TOcView* sein.

Öffentliche Elementfunktionen

ReleaseDataObject

```
uint32 ReleaseDataObject();
```

Enthält das Objekt *TOcInitInfo* einen Zeiger auf das Datenobjekt, aus dem der neue Teil erzeugt werden soll, dekrementiert *ReleaseDataObject* den Referenzzähler des Datenobjekts. Diese Funktion sollten Sie aufrufen, wenn Sie mit dem Datenobjekt fertig sind.

Siehe auch TOcInitInfo::Data

Aufzählungstyp TOcInitWhere ocf/ocobject.h

```
enum TOcInitWhere
```

Diese Werte geben an, wo sich die Daten für ein Objekt befinden. Ein Container übergibt diese Informationen an ein *TOcInitInfo*-Objekt, wenn er ein neues OLE-Objekt zum Verknüfen oder Einbetten erhält. Der Server kann jeden der verfügbaren Kanäle für die Übertragung der Daten in dem Objekt wählen.

Konstante	Bedeutung
iwFile	Der Server übergibt die Daten in einer Datei.
iwStorage	Der Server übergibt die Daten in einem Speicherabteil-Objekt (Teil einer Verbunddatei).
iwDataObject	Der Server übergibt die Daten in einem Datenübertragungsobjekt, das die Standard-OLE-Schnittstelle *IDataObject* unterstützt. (Über die Zwischenablage oder durch Ziehen übertragene Objekte untertützen diese Schnittstelle. *TOcInitInfo* enthält einen Zeiger auf die Schnittstelle.)
iwNew	Der Server wird aufgefordert, ein neues Objekt zu erzeugen.
iwHandle	Der Server übergibt ein Speicher-Handle für die Daten.

Siehe auch Öffentliche Konstruktoren TOcInitInfo, Aufzählungstyp TOcInitHow

Aufzählungstyp TOcInvalidate ocf/ocobject.h

enum TOcInvalidate

Funktionen, die ein Objekt für ungültig erklären, verwenden diese Aufzählungswerte für die Angabe, ob sich die Daten in einem Objekt geändert haben, oder ob sich das Erscheinungsbild eines Objekts geändert hat. Die Daten in einem Objekt können sich ändern, ohne daß die View des

Objekts für ungültig erklärt wird. Beispielsweise führt die Bearbeitung von Daten nicht notwendigerweise zu einer Aktualisierung der View, wenn das Objekt als Symbol gezeichnet wurde. Wenn sich sowohl die Daten wie auch die View ändern, sollten beide Flags mit dem bitweisen Operator OR (|) kombiniert werden.

Ist die View nicht mehr gültig, muß das Objekt neu gezeichnet werden. Sind die Daten ungültig, muß das Objekt gespeichert werden. (Es ist nicht nötig, das Objekt sofort zu sichern. *invData* gibt einfach nur an, daß das Objekt dirty ist und gespeichert werden muß, bevor das Dokument geschlossen wird.

Konstante	Bedeutung
invData	Die Daten in einem Objekt haben sich geändert und sollten im Container aktualisiert werden.
invView	Das Erscheinungsbild eines Objekts muß sich ändern und sollte im Container aktualisiert werden.

Siehe auch TOcRemView::InvalidateTOc, TOleView::InvalidatePart, TOleWindow::InvalidatePart

Struktur TOcMenuDescr ocf/ocapp.h

Die Menüdeskriptor-Struktur wird verwendet, wenn die Menüs eines Containers und eines Servers für das direkte Bearbeitung im Container-Dokument gemeinsam benutzt werden. Die Struktur enthält ein Handle auf ein gemeinsam benutztes Windows-Menüobjekt und die Anzahl der Dropdown-Menüs in jeder Gruppe.

Beim Arbeiten mit ObjectWindows können die Informationen in der Struktur dazu verwendet werden, ein *TMenuDescr*-Objekt für die andere Anwendung zu erzeugen. Um zwei Menüs gemeinsam zu nutzen, muß *TMenuDescr::Merge* aufgerufen werden. Wird ObjectWindows nicht eingesetzt, können die Windows-API-Routinen wie beispielsweise *InsertMenu* aufgerufen werden, um eigene Befehle in das gemeinsam benutzte Menü einzusetzen.

Die folgenden Botschaften haben ein *TOcMenuDescr* **struct** in ihren *IParams*: OC_APPINSMENUS, OC_APPMENUS und OC_VIEWINSMENUS. Die OLE-fähigen Fenster und View-Klassen von ObjectWindows verarbeiten diese Botschaften. Solange nicht ohne ObjectWindows programmiert wird, muß *TOcMenuDescr* normalerweise nicht direkt angesprochen werden. Beispiele zur Verarbeitung dieser Botschaften finden sich im Quellcode für die relevanten Ereignisbehandlungsroutinen in *TOleView, TOleWindow, TOleFrame* und *TOleMDIFrame*.

Siehe auch OC_APPxxxx-Botschaften, OC_VIEWxxxx-Botschaften, Klasse TMenuDescr

Öffentliche Datenelemente

HMenu

HMENU HMenu;
Enthält ein Handle für das gemeinsam genutzte Menü. Das Handle ist nur solange gültig, wie das Menü konstruiert wird. Nicht für weitere Verwendung speichern.

Width

int Width[6];
Das Array *Width* enthält die Anzahl der Popup-Menüs in jeder Menügruppe. Die Gruppen sind Datei, Bearbeiten, Container, Objekt, Fenster und Hilfe - in dieser Reihenfolge.

Das Array soll dabei helfen, ein *TMenuDescr*-Objekt zu konstruieren. Die Zahlen, die es enthält, steuern, wie das Menü gemeinsam benutzt wird.

Siehe auch Öffentliche Konstruktoren und Destruktoren TMenuDescr

Klasse TOcModule ocf/ocapp.h

TOcModule ist eine gemischte Klasse zur Ableitung von OLE-fähigen Anwendungsklassen. Jede ObjectComponents-Anwendung, die Verknüpfung und Einbettung unterstützt, sollte ihre Anwendungsklasse sowohl aus *TApplication* wie auch aus *TOcModule* ableiten. Die Modulklasse von ObjectComponents koordiniert einige Grundfunktionen der Registrierung und der Speicherverwaltung. Sie enthält außerdem einen Zeiger auf das Objekt *TOcApp*, das das Anwendungsobjekt durch ObjectComponents an OLE anbindet. *TOcModule* diese Arbeiten ausführen zu lassen macht es auch einfacher, den gleichen Programmcode sowohl für .EXE- wie auch für .DLL-Versionen des gleichen Servers zu verwenden.

Siehe auch Klasse TApplication, Klasse TOcApp

Öffentlicher Konstruktor und Destruktor

Konstruktor

TOcModule();
Erzeugt ein *TOcModule*. Nach der Erzeugung eines *TOcModule*-Objekts muß *OcInit* aufgerufen werden.

Destruktor

~TOcModule();
Gibt das *TOcApp*-Objekt frei. Eine Anwendung, die sich von *TOcModule* ableitet, muß die Methode *TOcApp::ReleaseObject* vor dem Schließen nicht aufrufen. (**delete** darf niemals aufgerufen werden, um ein *TOcApp*-Objekt zu entfernen.)

Siehe auch TOcModule::OcInitTOcModuleOcInit

Öffentliche Elementfunktionen

GetRegistrar

TRegistrar& GetRegistrar();
Gibt das Registrar-Objekt der Anwendung zurück. *OcInit* muß vorher aufgerufen worden sein.

Siehe auch TOcModule::OcInit, Klasse TRegistrar

IsOptionSet

bool IsOptionSet(uint32 option) const;
Gibt **true** zurück, wenn das Kommadozeilen-Flag, das durch *option* angegeben wird, gesetzt ist und **false**, wenn es nicht gesetzt ist. Der Registrar setzt die Flags, wenn er auf OLE bezogene Schalter auf der Kommandozeile der Anwendung auswertet. Die möglichen Werte für *option* werden in *TOcAppMode* aufgezählt.

Siehe auch Klasse TOcAppMode, Klasse TOcRegistrar

OcInit

void OcInit(TOcRegistrar& registrar, uint32 options);
Initialisiert die ObjectComponents-Unterstützung für das Code-Modul. Dieser Aufruf veranlaßt ObjectComponents, das *TOcApp*-Verbinderobjekt zu erzeugen, das die Anwendung an das OLE-System anbindet. *OcInit* muß immer direkt nach dem Erzeugen des Modul-Objekts aufgerufen werden.

registrar ist das Registrar-Objekt der Anwendung. Es muß erzeugt werden, bevor *OcInit* aufgerufen wird.

options ist eine Menge von Bitflags, die Kommanozeilen-Optionen, die für diese Instanz des Programms gesetzt wurden, beschreiben. *IsOptionSet* kann aufgerufen werden, um bestimmte Optionen zu testen. Die möglichen Options-Flags werden in *TOcAppMode* definiert.

Siehe auch Klasse TOcApp, Aufzählungstyp TOcAppMode, TOcModule::IsOptionSet, Klasse TOcRegistrar

Öffentliche Datenelemente

OcApp

TOcApp* OcApp;
Enthält das *TOcApp*-Objekt, das das Partnerobjekt der von *TApplication* abgeleiteten Klasse ist. Dieses Element wird bei Aufruf von *OcInit* initialisiert.

Siehe auch Klasse TOcApp, TOcModule::OcInit

OleMalloc

TOleAllocator OleMalloc;
Erzeugt ein Zuweisungsobjekt, das das OLE-System initialisiert und erzeugt den Speicherplatzreservierer. OLE ermöglicht es jedem Programm, eine Speicherverwaltung für OLE auzubauen, die bei der Speicherzuweisung und -entfernung für die betreffende Anwendung verwendet wird.

TOcModule wählt einfach den Standard-Speicherplatzreservierer aus. Sollten Sie ungewöhnliche Anforderungen an die Speicherverwaltung stellen und daher Ihre eigene, benutzerdefinierte Speicherzuweisung vornehmen wollen, setzen Sie bitte seine *IMalloc*-Schnittstelle in *OleMalloc::Mem*.

Siehe auch Klasse TOleAllocator, TOleAllocator::Mem

Klasse TOcNameList ocf/ocapp.h

TOcApp verwendet diese Klasse intern, um eine Kollektion von *TOcFormatName*-Objekten zu verwalten. Jedes Formatnamenobjekt enthält drei Strings, die ein Datenformat der Zwischenablage beschreiben - beispielsweise Text oder Bitmap. *TOcApp* zeigt diese Strings in Standard-OLE-Dialogfeldern wie Einfügen und Verknüpfen an.

Die Liste der Formatnamen wird innerhalb von *TOcApp* erzeugt und verwaltet. Normalerweise muß diese Liste nicht direkt angesprochen werden. Sollen jedoch benutzerdefinierte Formate in die Liste aufgenommen werden, müssen diese registriert werden. Vergleiche *TOcApp::AddUserFormatName*. Dort finden sich weitere Informationen zur Erstellung von benutzerdefinierten Formaten.

Standard-Windows-Formate der Zwischenablage werden der Liste immer automatisch hinzugefügt. Der Name und die Ergebnis-Strings für Standardformate werden in OLEVIEW.RC definiert. Bearbeiten Sie diese

Datei, um die Strings zu lokalisieren. (Standardformate haben keinen Bezeichner-String, sondern eine Registriernummer wie beispielsweise CF_TEXT.)

Siehe auch

Klasse TOcApp, TOcApp::AddUserFormatName, Klasse TOcFormatName

Öffentliche Konstruktoren und Destruktoren

Konstruktor

TOcNameList();
Erzeugt eine Namensliste, die keine Einträge enthält. Zum Einfügen von Namen in die Liste muß *Add* aufgerufen werden.

Destruktor

~TOcNameList();
Entfernt die Liste und die Objekte in der Liste.

Siehe auch

TOcNameList::Add

Öffentliche Elementfunktionen

operator []

Form 1

TOcFormatName*& operator[](unsigned index);
Gibt aus der Liste der Formatnamen-Objekte den Eintrag an der Position *index* zurück. Das erste Objekt befindet sich bei Index 0. Zeigt *index* hinter das Ende der Liste, löst die Funktion eine Vorbedingungs-Exception aus.

Form 2

TOcFormatName* operator[](char far* id);
Gibt das Formatnamen-Objekt zurück, dessen Format-ID-String *id* entspricht. Der Rückgabewert ist 0, wenn keine Entsprechung gefunden wurde.

Siehe auch

TOcFormatName

Add

int Add(TocFormatName* name);
Fügt das Objekt *name* der Liste hinzu. Gibt bei Erfolg 1, bei Mißerfolg 0 zurück.

Siehe auch

TocNameList::Clear, TocNameList::Detach

Clear

void Clear(int del = 1);
Leert die Liste. Wenn *del* 1 ist, löscht *Clear* außerdem alle Objekte in der Liste.

Siehe auch TOcNameList::Add, TOcNameList::Detach

Count

virtual uint Count() const;
Gibt die Anzahl der Einträge in der Liste zurück.

Siehe auch TOcNameList::IsEmpty

Detach

int Detach(const TOcFormatName* name, int del = 0);
Entfernt das einzelne Objekt *name* aus der Liste. Wenn *del* 1 ist, löscht *Detach* auch das Objekt *name*.

Siehe auch TOcNameList::Clear, TOcNameList::Add

Find

unsigned Find(const TOcFormatName* name) const;
Durchsucht die Liste und gibt die Position von *name* zurück. Wenn das *name*-Objekt sich nicht in der Liste befindet, gibt *Find* UINT_Max zurück.

IsEmpty

int IsEmpty() const;
Gibt 1 zurück, wenn die Liste gegenwärtig keine Einträge enthält und 0, wenn die Liste mindestens einen Eintrag enthält.

Siehe auch TOcNameList::Count

Klasse TOcPart ocf/ocpart.h

Basisklasse

TUnknown
Ein *TOcPart*-Objekt stellt ein verknüpftes oder eingebettetes Objekt in einem Dokument dar. Es stellt das verknüpfte oder eingebettete Objekt so dar, wie es der Container betrachtet. Aus der Sicht des Servers besteht das

gleiche verknüpfte oder eingebettete OLE-Objekt aus zwei Teilen: aus Daten (*TOcDocument*) und aus der grafischen Darstellung der Daten (*TOcRemView*). *TOcPart* verwaltet eine Stelle im Dokument des Containers, an der ein Server ein OLE-Objekt ablegt.

*TOcPart*ist ein COM-Objekt und implementiert die Schnittstelle *IUnknown*.

Siehe auch

Klasse TOcDocument, Klasse TOcPartCollection, Klasse TOcRemView, Klasse TUnknown

Öffentliche Konstruktoren

Konstruktoren

Form 1

TOcPart(TOcDocument& document, TOcInitInfo far& initInfo, TRect pos, int id = 0);

document ist das Objekt *TOcDocument* des Containers und stellt das Verbunddokument dar, das den neu erzeugten Teil enthalten wird. *initInfo* enthält Informationen über das Objekt, das eingefügt wird und wird normalerweise während einer Einfügeoperation (paste, drop oder insert) erzeugt. Die Koordinaten in *pos* legen den Bereich fest, in dem das neue Objekt gezeichnet wird. *id* ist jeder frei wählbare, eindeutige Integerwert, der dazu dient, dieses Objekt von anderen im gleichen Dokument zu unterscheiden. Wenn *id* 0 ist, generiert *TOcPart* automatisch eine neue ID.

Form 2

TOcPart(TOcDocument& document, const char far* name);

Für *document* gilt das gleiche wie in Form 1. Der String *name* ist der Name eines verknüpften oder eingebetteten Teils. Die zweite Form wird dann verwendet, wenn ein Teil aus einem Verbunddokument geladen wird. Der Name des Teils ist auch der Name des Speicherabteils, in den der Teil geschrieben wurde.

Beide Konstruktoren erwarten, das eigene *TOcDocument*-Objekt des Containers zu erhalten. Es repräsentiert das Verbunddokument, das das neue Objekt aufnimmt.

Siehe auch

Klasse TOcDocument, Klasse TOcInitInfo, TOcPart::Delete, Klasse TRect

Öffentliche Elementfunktionen

operator ==

bool operator==(const TOcPart& other);

Gibt **true** zurück, wenn *other* der gleiche *TOcPart* wie **this** ist. Dieser Operator ist für die Verwendung der Klasse *TOcPartCollection* definiert.

Activate

bool Activate(bool activate);
Wenn *activate* **true** ist, aktiviert diese Funktion den Teil durch Aufforderung des Servers, seine primäre (oder Standard-) Aktionsart für das Objekt auszuführen. Wenn die Standard-Aktionsart beispielsweise *Edit* ist, beginnt *Activate* eine Bearbeitungssitzung an Ort und Stelle. Wenn *activate* **false** ist, deaktiviert diese Funktion eine Bearbeitungssitzung an Ort und Stelle.

Activate gibt **true** zurück, wenn der Server den Befehl ausführen kann.

Siehe auch TOcPart::IsActive, TOcPart::Open

Close

bool Close();
Löst die Verbindung des eingebetteten Objekts zum Server. Gibt **true** zurück, wenn der Server erfolgreich geschlossen wird.

Delete

void Delete();
Delete wird verwendet, wenn der Anwender ein eingebettetes Objekt auswählt und die Entfernen-Taste drückt (oder eine Ausschneiden-Operation durchführt). Es ruft zuerst *Close* auf, um die Verbindung des Containers zum eingebetteten Objekt zu lösen. Dann gibt es die Referenz auf den eingebetteten Teil frei.

Siehe auch TOcPart::Close

Detach

int Detach();
Trennt einen Teil von seinem Dokument. *Detach* soll beispielsweise vor dem Ausschneiden eines Teils für die Zwischenablage aufgerufen werden.

DoVerb

bool DoVerb(uint whichVerb);
Weist den Server an, einen seiner Befehle auf den Teil auszuführen. Eine Aktionsart ist normalerweise eine Aktion wie Editieren oder Ausführen. Ein Server kann mehrere Aktionsarten unterstützen. *whichVerb* identifiziert eine bestimmte Aktionsart durch ihren Ordinalwert. (Die erste Aktionsart, also die primäre oder Standard-Aktionsart, ist Null.) *DoVerb* gibt **true** zurück, wenn es dem Server gelingt, die angeforderte Aktion zu be-

enden. Die Ausführung einer Aktionsart kann den Teil veranlassen, aktiviert zu werden.

Siehe auch TOcPart::EnumVerbs

Draw

bool Draw(HDC dc, const TRect& pos, const TRect& clip, TOcAspect aspect = asDefault);
Zeichnet den Teil auf dem Bildschirm. Wurde der Teil noch nicht geladen, lädt ihn *Draw* zuerst. *dc* ist ein Windows-Gerätekontext, in den der Teil gezeichnet werden muß. Die Koordinaten in *pos* geben an, wo im Fenster der Teil abgelegt werden soll. Außerhalb des durch das Rechteck *clip* bestimmten Bereichs kann der Server nicht zeichnen. *clip* und *pos* konnen gleich sein. Wenn *clip* ein leeres Rechteck beschreibt, kann der Server überall zeichnen. *aspect* steuert, wie die Daten präsentiert werden - beispielsweise als Symbol.

Siehe auch Aufzählungstyp TOcAspect, Klasse TRect

EnumVerbs

bool EnumVerbs(const TOcVerb& verb);
EnumVerbs wird aufgerufen, um die Aktionsarten festzustellen, die ein Server für einen bestimmten Teil unterstützt. Jeder Aufruf von *EnumVerbs* setzt eine andere Aktionsart in den Parameter *verb*. *EnumVerbs* gibt **false** zurück, wenn alle Aktionsarten des Servers augezählt wurden.

TOleWindow ruft *EnumVerbs* auf, um Aktionsarten für das aktive Objekt auf das Menü Bearbeiten des Containers zu setzen.

GetName

LPCOLESTR GetName();
Gibt den String zurück, der den Teil identifiziert. Jeder Teil eines Dokuments besitzt einen anderen Namen. ObjectComponents erzeugt die Namen automatisch durch Inkrementierung einer internen ID-Nummer für jeden neuen Teil.

Siehe auch TOcPart::GetNameLen, TOcPart::Rename

GetNameLen

int GetNameLen();
Gibt die Anzahl der Zeichen im Namensstring, der den Teil kennzeichnet, zurück. Diese Zahl berücksichtigt die abschließende Null nicht.

Siehe auch TOcPart::GetName, TOcPart::Rename

GetPos

TPoint GetPos() const;
Gibt die Position des Teils innerhalb seines Container-Dokuments zurück. Die Position gibt die linke, obere Ecke des Teils in Client-Bereichskoordinaten an. Die Koordinaten berücksichtigen jede Skalierung, die für das *TOcView*-Objekt, das den Teil enthält, gesetzt wurde.

Siehe auch TOcPart::GetRect, TOcPart::GetSize, TOcPart::SetPos, Klasse TPoint

GetRect

TRect GetRect() const;
Gibt das Rechteck zurück, das das Bild des Teils im Client-Bereich des Containers begrenzt. Die Position des Rechtecks wird in Client-Bereichskoordinaten angegeben.

Siehe auch TOcPart::GetPos, TOcPart::GetSize, TOcPart::UpdateRect, Klasse TRect

GetServerName

LPCOLESTR GetServerName(TOcPartName partName);
Fragt bei OLE den Namen des Objekts oder des Objekt-Servers an - abhängig von *partName*. Ein Conatiner kann diese Information in seiner Titelleiste anzeigen.

Diese Funktion wird in der aktuellen Implementierung von ObjectWindows nicht verwendet. Das Objekt *TOcView* automatisiert den Fenstertitel des Containers automatisch.

Siehe auch Aufzählungstyp TOcPartName

GetSize

TSize GetSIze() const;
Gibt die Bildgröße des Teils im Container-Dokument zurück. Die Felder des Rückgabewertes geben die Breite und die Höhe des Teils in Client-Bereichskoordinaten an. Die Koordinaten berücksichtigen Skalierung.

Siehe auch TOcPart::GetPos, TOcPart::GetRect, TOcPart::SetSize, Klasse TSize

IsActive

bool IsActive() const;
Gibt **true** zurück, wenn der Teil gegenwärtig aktiv ist. Wenn nicht, wird **false** zurückgegeben.

Siehe auch TOcPart::SetActive

IsLink

bool IsLink() const;
Gibt **true** zurück, wenn der Teil ein verknüpftes OLE-Objekt darstellt und **false**, wenn es sich um ein eingebettetes OLE-Objekt handelt. Ein Container kann diese Methode verwenden, um visuell zwischen verknüpften und eingebetteten Objekten zu unterscheiden. Der Quellcode für *TOleWindow::PaintParts* enthält ein Beispiel hierfür.

IsSelected

bool IsSelected() const;
Gibt **true** zurück, wenn der Teil gegenwärtig ausgewählt ist. Wenn nicht, wird **false** zurückgegeben. Diese Funktion wird laufend in Schleifen, die alle ausgewählten Teile in einem Dokument verarbeiten, aufgerufen. Beispielsweise ruft *TOleView* beim Zeichnen der Teile in einem Dokument *IsSelected* für jedes Teil auf, um festzustellen, wo die Auswahlfelder gezeichnet werden müssen.

Statusinformationen zur Auswahl werden ausschließlich in *TOcPart* verwaltet. Sie beeinflussen das OLE-Objekt selbst nicht.

Siehe auch TOcPart::Select

IsVisible

Form 1

bool IsVisible() const;
Gibt **true** zurück, wenn der Teil gegenwärtig sichtbar ist und **false**, wenn er verborgen ist.

Form 2

bool IsVisible(const TRect& logicalRect) const;
Gibt **true** zurück, wenn der Teil gegenwärtig im vorgegebenen Bereich *logicalRect* des Container-Fensters sichtbar ist. Gibt **false** zurück, wenn der Teil nicht sichtbar ist, beispielsweise weil der Anwender einen Bildlauf in einen anderen Teil des Dokuments durchgeführt hat.

Siehe auch TOcPart::SetVisible

Load

bool Load();
Initialisiert ein *TOcPart*-Objekt mit den aus dem Speicherabteil ausgelesenen Informationen.

Siehe auch TOcPart::Save

Open

bool Open(bool open);
Wenn *Open* **true** ist, veranlaßt der Befehl *Open* den Server, eine Editiersitzung im Server-Dokument zu beginnen. Genauer gesagt: Der Server wird augefordert, seinen Kontext Öffnen auszuführen. Wenn *Open* **false** ist, weist der Befehl den Server an, sein geöffnetes Editierfenster zu verbergen. Die Sitzung wird jedoch nicht beendet. Bei Erfolg gibt *Open* **true** zurück. *Open* gibt **false** zurück, wenn der Server Editieren nicht unterstützt.

Note *TOcPart::Close* ist nicht das Gegenteil von *TOcPart::Open*. **false** muß an *Open* übergeben werden, um das Editieren zu beenden.

Siehe auch TOcPart::Activate

Rename

void Rename();
Veranlaßt den Teil, den internen Namen, den ObjectComponents zur Unterscheidung der Teile in einem Dokument generiert, zu aktualisieren. Rufen Sie *Rename* immer auf, wenn Sie die Dokumentdatei umbenennen. OLE verwendet bei der Erzeugung von Verknüpfungen den Namen des Objekts. Daher muß der Objektname den Dateinamen genau wiederspiegeln, damit die Verknüpfungen funktionieren.

Siehe auch TOcPart::GetName, TOcPart::GetNameLen

Save

Form 1 **bool Save(bool sameAsLoaded = true);**
Veranlaßt den Teil, sich in den Datei-Stream des Dokuments zu schreiben. Wenn *sameAsLoaded* **true** ist, speichert sich der Teil im gleichen Speicherabteil, in das er zuletzt geschrieben wurde. Wird *sameAsLoaded* auf **false** gesetzt, veranlaßt dies den Teil, sich unter dem neuen Stammspeicherabteil des Dokuments ein neues Speicherabteil zu erzeugen. Normalerweise sollte *sameAsLoaded* als Antwort auf einen Datei | Speichern Befehl **true** sein und **false** als Antwort auf Datei | Speichern als.

(Ein Speicherabteil ist ein Abteil in einer Verbunddatei. ObjectComponents verwaltet die Speicherabteile. Normalerweise müssen keine ausdrücklichen Befehle abgesetzt werden, wo Teile zu speichern sind.)

Form 2

bool Save(Istorage* storage, bool sameAsLoaded, bool remember);
Die zweite Form akzeptiert einen Zeiger auf eine *Istorage*-Schnittstelle. Damit kann gesteuert werden, wohin ein Objekt gespeichert wird. Für *sameAsLoaded* gilt das gleiche wie in Form 1. *remember* weist den Teil an, ob er sich an das Objekt in *storage* erinnern soll. Wenn man einen Teil in seiner üblichen Datei speichert, soll er sich normalerweise an sein eigenes Speicherabteil erinnern. Will man hingegen einen Teil kopieren, soll er normalerweise sein ursprüngliches Speicherobjekt behalten und nicht dasjenige, in das die Kopie gespeichert wird. Beispielsweise sollte *remember* **false** sein, wenn man eine Kopie in eine Datei für die Zwischenablage speichert.

Siehe auch TOcPart::Load

Select

void Select(bool select);
Teilt dem Teil mit, ob er aktuell ausgewählt ist. Zum Auswählen wird *select* auf **true** gesetzt und zum Aufheben der Auswahl auf **false**. Der Anwender wählt Objekte aus, um Operationen mit ihnen durchzuführen. Beispielsweise wählt der Anwender ein Objekt aus, bevor er es in die Zwischenablage kopiert. Beim Zeichnen seiner Teile fragt *TOleView* jedes einzelne ab und zeichnet ein Auswahlfenster um jedes, das der Anwender ausgewählt hat.

Siehe auch TOcPart::IsSelected

SetActive

void SetActive();
Synchronisiert ein internes Flag mit dem aktuellen Status des Objekts - aktiv oder inaktiv. Normalerweise sollte diese Funktion nicht aufgerufen werden müssen. Rufen Sie an ihrer Stelle *TOcPart::Activate* auf, um den Teil zu aktivieren.

Siehe auch TOcPart::Activate, TOcPart::IsActive

SetHost

bool SetHost(IBContainer far* container);
Verschiebt den Teil von einem Container zu einem anderen. *container* kann ein Objekt vom Typ *TOcView* (oder ein daraus abgeleitetes Objekt)

sein. Es bestimmt die View, die den Teil erhält. *SetHost* wird normalerweise nicht aus der Anwendung aufgerufen.

IBContainer ist eine benutzerdefinierte Schnittstelle, die innerhalb der Support-Bibliothek BOCOLE definiert ist. *TOcView* implementiert diese Schnittstelle.

SetPos

void SetPos(const TPoint& pos);
Weist die Position des Teils innerhalb des Container-Dokuments zu. Die Position bestimmt die obere, linke Ecke des Teils in Pixeln. Es wird von der oberen, linken Ecke des Client-Fensters des Containers gemessen. Die Koordinaten berücksichtigen Skalierung.

Siehe auch TOcPart::GetPos, TOcPart::SetSize, TOcPart::UpdateRect, Klasse TPoint

SetSize

void SetSize(const TSize& size);
Legt die Bildgröße des Teils im Container-Dokument fest. *size* setzt die Breite und Höhe des Teils in Client-Bereichskoordinaten fest. Die Koordinaten berücksichtigen Skalierung, die für das *TOcView*-Objekt, das den Teil enthält, festgelegt wurde.

Siehe auch TOcPart::GetPos, TOcPart::SetSize, TOcPart::UpdateRect, Klasse TSize

SetVisible

void SetVisible(bool visible);
Zeigt oder verbirgt den Teil abhängig vom Wert von *visible*.

Siehe auch TOcPart::IsVisible

Show

bool Show(bool show);
Macht den Teil sichtbar. *show* wird verwendet, um die Verknüpfungsquelle aufzufordern, sich selbst im Container-Fenster zu zeigen. Wenn *show* **false** ist, verbirgt sich der Teil. Der Rückgabewert ist bei Erfolg **true**.

Siehe auch TOcPart::IsVisible

UpdateRect

void UpdateRect();
Ordnet den Teil einem neuen Rechteck zu, wenn seine Größe oder Position sich ändert. Wird von *SetPos* und *SetRect* aufgerufen.

Siehe auch TOcPart::GetRect, TOcPart:SetPos, TOcPart::SetSize

Geschützter Destruktor

Destruktor

~TOcPart();
Entfernt das Objekt *TOcPart*.

Klasse TOcPartCollection ocf/ocpart.h

Verwaltet eine Menge von *TOcPart*-Objekten. Jedes *TOcDocument* erzeugt ein Teile-Kollektionsobjekt, um die Menge der OLE-Objekte, die mit dem Dokument verknüpft oder in es eingebettet sind, zu verwalten. Das Teile-Kollektionsobjekt fügt Teile hinzu, löscht Teile, sucht sie, zählt sie und hilft dem Dokument allgemein, die Übersicht über seine Bestandteile zu behalten.

Da *TOcDocument* ein Teile-Kollektionsobjekt enthält, muß die Kollektion normalerweise nicht direkt erzeugt oder bearbeitet werden.

Siehe auch Klasse TOcPart, Klasse TOcPartCollectionIter

Öffentlicher Konstruktor und Destruktor

Konstruktor

TOcPartCollection();
Erzeugt eine leere Kollektion. Durch den Aufruf von *Add* können der Kollektion Teile hinzugefügt werden.

Destruktor

~TOcPartCollection();
Gibt alle Server frei, die verknüpfte oder eingebettete Objekte zur Verfügung stellen.

Öffentliche Elementfunktionen

Add

int Add(TOcPart* const& part);
Fügt der Kollektion einen neuen Teil hinzu. Gibt bei Erfolg 1 und bei Mißerfolg 0 zurück.

Siehe auch Klasse TOcPart

Clear

void Clear();
Löst die Verbindung aller Teile zu ihren Servern, entfernt sie aus der Kollektion und gibt sie frei. Teilt OLE mit, daß diese Kollektion keinen weiteren Bedarf für diese Server hat.

Count

virtual unsigned Count() const;
Gibt die Anzahl der Teile, die sich aktuell in der Kollektion befinden, zurück.

Detach

int Detach(TOcPart* const& part, int del = 0);
Entfernt *part* aus der Kollektion. Wenn *del* ungleich Null ist, gibt *Detach* auch das Objekt *TOcPart* frei. Wenn der interne Referenzzähler des Teils als Ergebnis Null erhält, löscht sich der Teil selbst. Gibt bei Erfolg 1 und bei Mißerfolg 0 zurück.

Siehe auch Klasse TOcPart

Find

unsigned Find(TOcPart* const& part) const;
Sucht nach *part* und gibt dessen Position in der Kollektion zurück. *Find* gibt UINT_MAX zurück, wenn sich *part* nicht in der Kollektion befindet.

Siehe auch Klasse TOcPart

IsEmpty

int IsEmpty() const;
Gibt **true** zurück, wenn die Kollektion gegenwärtig keine Objekte enthält und **false**, wenn sie wenigstens ein Objekt enthält.

Locate

TOcPart* Locate(TPoint& point);
Gibt das Teileobjekt zurück, das an einem bestimmten Punkt des Bildschirms sichtbar ist. Die Zahlen des Punktes werden als logische Koordinaten interpretiert. *Locate* gibt 0 zurück, wenn sich kein Teil der Kollektion an dem vorgegebenen Punkt befindet.

Siehe auch Klasse TPoint

SelectAll

bool SelectAll(bool select = false);
Legt den Auswahlstatus aller Teile der Kollektion fest. Wenn *select* **true** ist, wählt *SelectAll* sie alle aus. Wenn *select* **false** ist, wird die Auswahl für alle Teile aufgehoben. Der Anwender kann Aktionen ausführen (beispielsweise Ziehen, Löschen und Kopieren), die sich auf alle ausgewählten Objekte beziehen.

Der Container markiert ausgewählte Objekte üblicherweise jeweils durch das Zeichnen eines Rechtecks mit Ziehmarken (Griffe zum Bewegen des Rechtecks). Die Klasse *TOleWindow* führt dies in ObjectWindows-Programmen automatisch aus.

Klasse TOcPartCollectionIter ocf/ocpart.h

Ein Teile-Kollektions-Iterator zählt die in ein Verbunddokument eingebetteten Objekte auf.

Ein Verbunddokument kann viele verknüpfte und eingebettete Objekte enthalten. Innerhalb des Containers wird jedes Objekt durch ein Objekt vom Typ *TOcPart* dargestellt. *TOcDocument* erzeugt ein Kollektions-Objekt vom Typ *TOcPartCollection*, um alle Teile, die es enthält, zu verwalten. Das Kollektions-Objekt fügt der Kollektion Elemente hinzu oder entfernt sie aus ihr. *TOcDocument* erzeugt außerdem einen Teile-Kollektions-Iterator, um durch die aktuelle Liste seiner Teile zu iterieren. Ein Iterator zeigt auf ein Element in der Kollektion. Der Iterator kann inkrementiert werden, um die Kollektion durchzugehen. Der Iterator teilt mit, wenn das Ende erreicht ist (der Operator ++ gibt 0 zurück).

Die Kollektion und ihr Iterator zusammen erlauben den Dokument eine große Flexibilität in der Verwaltung seiner Objekte.

Siehe auch Klasse TOcPart, Klasse TOcPartCollection

Öffentlicher Konstruktor

Konstruktor

TOcPartCollectionIter(const TOcPartCollection& coll);
Erzeugt einen Iterator, der die in der Kollektion *coll* enthaltenen Objekte aufzählt.

Siehe auch Klasse TOcPartCollection

Öffentliche Elementfunktionen

operator ++

Form 1

TOcPart* operator++(int);
Gibt den aktuellen Teil zurück und setzt den Iterator so herauf, daß er auf den nächsten Teil zeigt (Postinkrement);

Form 2

TOcPart* operator++();
Inkrementiert den Iterator, so daß er auf den nächsten Teil in der Liste zeigt und gibt dann diesen Teil zurück (Preinkrement).

Current

TOcPart Current() const;
Gibt den Teil zurück, auf den der Iterator gegenwärtig zeigt.

operator int

operator int() const;
Kovertiert den Iterator in einen Integerwert und testet so, ob der Iterator die Aufzählung der Kollektion beendet hat. Gibt Null zurück, wenn der Iterator das Ende der Liste erreicht hat und einen Wert ungleich Null, wenn dies nicht der Fall ist.

Restart

Form 1

void Restart();
Setzt den Iterator zurück, damit er wieder mit dem ersten Teil im Dokument beginnt.

Form 2

void Restart(unsigned start, unsigned stop);
Setzt den Iterator zurück, damit er einen Teilbereich von Objekten im Dokument aufzählt. Er beginnt in der Liste mit dem Objekt an der Position *start* und hört mit dem Objekt an der Position *stop* auf.

Aufzählungstyp TOcPartName ocf/ocobject.h

enum TOcPartName
Wenn der Container den Namen eines Teils beim Server anfragt, kann es sich dabei um jede beliebige der möglichen Antworten handeln. Diese Werte geben an, welchen Namen der Container als Antwort erhalten möchte.

Konstante	Bedeutung
pnLong	Der String, den der Server als *description* für diesen Objekttyp registriert hat.
pnShort	Der String, den der Server als *progid* für diesen Objekttyp registriert hat.
pnApp	Der String, den der Server als *description* für die Server-Anwendung insgesamt registriert hat.

Siehe auch Beschreibung Registrierschlüssel, progid Registrierschlüssel, TOcPart::GetServerName

Klasse TOcRegistrar ocf/ocapp.h

Basisklasse

TRegistrar
TOcRegistrar verwaltet alle Registrierungsaufgaben einer Anwendung. Es verarbeitet auf OLE bezogene Schalter auf der Kommandozeile und zeichnet notwendige Informationen zu der Anwendung in der Registrierungsdatenbank des Systems auf. Wurde die Anwendung bereits in der Datenbank registriert, bestätigt der Registrar, daß die registrierten *path, progid* und *clsid* noch stimmen. Ist dies nicht der Fall, wird die Anwendung neu registriert.

Jede Anwendung unter ObjectComponents muß ein Registrar-Objekt erzeugen. Erzeugen Sie ein *TOcRegistrar-Objekt*, wenn Ihre Anwendung Verknüpfung und Einbettung unterstützt. Wenn Ihre Anwendung Automatisierung, nicht aber Verknüpfung und Einbettung unterstützt, sollten Sie statt dessen ein *TRegistrar*-Objekt erzeugen. *TOcRegistrar* erweitert *TRegistrar* durch Anbindung der Anwendung an die Schnittstellen der Support-Bibliothek BOCOLE, die Verknüpfung und Einbettung unterstützen.

Die Hauptprozedur einer Anwendung führt normalerweise die folgenden Aktionen mit ihrem Registrar aus:

Erzeugen des Registrars, Übergabe an einen Zeiger auf den Factory-Callback der Anwendung.

Aufrufen von *IsOptionSet* auf, um die Optionen zu prüfen, die möglicherweise die Art des Starts der Anwendung beeinflussen (beispielsweise unsichtbar zu bleiben, wenn der Aufruf für eine Einbettung geschieht).

Aufrufen von *Run*, um in die Meldungsschleife des Programms einzutreten.

TOcRegistrar erbt sowohl *IsOptionSet* wie auch *Run* von seiner Basisklasse *TRegistrar.*

Siehe auch

Registrierschlüssel clsid, Registrierschlüssel path, Registrierschlüssel progid, Klasse TRegistrar

Öffentlicher Konstruktor und Destruktor

Konstruktor

TOcRegistrar(TRegList& regInfo, TComponentFactory callback, string& cmdLine, HINSTANCE hInst = _hInstance);
regInfo ist die Registrationsstruktur der Anwendung (üblicherweise wird der Name *appReg* vergeben).

callback ist die Funktion Factory-Callback, die ObjectComponents aufruft, wenn die Anwendung ein Objekt erstellen muß. Ein ObjectWindows-Programm kann die Klasse *TOleFactory* verwenden, um diesen Callback zu implementieren.

cmdLine einthält den Kommandozeilen-String, der die Anwendung aufgerufen hat.

hInst ist die Instanz der Anwendung.

Destruktor

~TOcRegistrar();
Entfernt die Objekte, die der Registrar intern verwendet.

Siehe auch

Typdefinition TComponentFactory, TOleFactory<>Klasse

Öffentliche Elementfunktionen

BOleComponentCreate

HRESULT BOleComponentCreate(IUnknown far* far* retIface, IUnknown far* outer, BCID idClass);
Ruft die Support-Bibliothek BOCOLE auf, um eines der Hilfsobjekte zu erzeugen, die ObjectComponents intern verwendet. Normalerweise muß *BOleComponentCreate* nicht direkt angesprochen werden.

- *retIface* nimmt eine Schnittstelle zu der angefragten Komponente auf.

- *outer* ist die Schnittstelle *IUnknown* des äußeren Objekts, von dem die neue Komponente ein Teil werden soll.
- *idClass* kennzeichnet diejenige Komponente, die erzeugt werden soll. Die möglichen Werte werden als *cidBolexxxx*-Konstanten in ocf/bole-defs.h definiert.

Der Rückgabewert ist ein OLE-Ergebnis: entweder HR_OK bei Erfolg oder HR_FAIL bei Mißerfolg.

Siehe auch

Ergebnismakros HR_xxxx

CreateOcApp

void CreateOcApp(uint32 options, TOcApp*& ref);

Erzeugt das Verbinderobjekt, das eine Anwendung an OLE anbindet. *options* ist eine Menge von Bitflags, die den Laufmodus der Anwendung angibt. Die möglichen Options-Flags sind in *TOcAppMode* definiert. *ret* gibt an, wo *CreateOcApp* einen Zeiger auf das neu erzeugte *TOcApp*-Verbinderobjekt setzt.

CreateOcApp wird während *TOcModule::OcInit* aufgerufen. Es sollte nicht direkt angesprochen werden müssen.

- *CreateOcApp* dient dazu, dem Anwender die Einzelheiten des Verbinderobjekts *TOcApp* zu ersparen. *TOcApp* hängt eng mit der Implementierung von ObjectComponents zusammen. Die Details der Initialisierung einer OLE-Sitzung können sich ändern.

Siehe auch

Klasse TOcApp, Aufzählungstyp TOcAppMode, TOcModule::OcInit

GetAppDescriptor

TAppDescriptor& GetAppDescriptor();

Gibt den Deskriptor der Anwendung zurück. ObjectComponents verwendet intern einen Anwendungsdeskriptor, der Informationen über ein Modul enthält. (Eine DLL erhält einen eigenen Anwendungsdeskriptor.) *TAppDescriptor* ist undokumentiert, da er nur intern verwendet wird und Änderungen unterliegt. Die Registrar-Klassen *TOcRegistrar* und *TRegistrar* stellen die unterstützten Schnittstellen zum Anwendungsdeskriptor dar. Der Registrar erzeugt den Deskriptor, und die meisten seiner Elementfunktionen rufen Deskriptor-Funktionen auf, um ihre Aufgaben durchzuführen.

Normalerweise muß diese Methode nicht direkt aufgerufen werden.

Geschützte Elementfunktionen

CanUnload

bool CanUnload();
Gibt **true** zurück, wenn die Anwendung gegenwärtig keinen OLE-Clients als Server dient. Andernfalls wird **false** zurückgegeben.

GetFactory

void far* GetFactory(const GUID& clsid, const GUID far& iid);
Gibt einen Zeiger auf die Factory-Schnittstelle zurück, damit der durch *clsid* angegebene Objekttyp erstellt werden kann. *iid* nennt diejenige Schnittstelle, die man erhalten möchte. Wenn der Registrar keine *iid*-Schnittstelle für *clsid*-Objekte finden kann, gibt er Null zurück.

ObjectComponents ruft bei jedem Laden einer DLL durch einen neuen Client das Element *GetFactory* der DLL auf. Normalerweise muß *GetFactory* nicht direkt angesprochen werden.

LoadBOle

void LoadBOle();
Lädt und intialisiert die Support-Bibliothek von ObjectComponents (BOCOLE.DLL). *LoadBOle* löst eine *TXObjComp*-Exception aus, wenn BOCOLE.DLL nicht gefunden wird oder wenn die installierte Version nicht mit der Version der Bibliothek, die die Anwendung benutzt, kompatibel ist.

Klasse TOcRemView — ocf/ocremvie.h

Basisklasse

TOcView
Ein Server zum Verknüpfen und Einbetten erzeugt ein entferntes Objekt, um sein OLE-Objekt im Container-Fenster zu zeichnen. *TOcRemView* zeichnet das Objekt nur. Um die Daten in dem Objekt zu laden und zu speichern, muß der Server auch ein *TOcDocument*-Objekt erzeugen. Das Dokument und die entfernte View stellen zusammen aus Sicht des Servers ein OLE-Objekt dar.

Der Container erzeugt ein *TOcPart*-Objekt für jedes OLE-Objekt, das er empfängt. Das Teil-Objekt des Containers kommuniziert durch OLE mit den Dokument- und View-Objekten des Servers. Der Teil übermittelt der View des Servers, wann und wo sie das Objekt zeichnen soll. Er teilt

außerdem dem Dokument des Servers mit, wann und wo das Objekt geladen oder gespeichert werden soll.

Die beiden Arten einer View, *TOcView* und *TOcRemView* dürfen nicht verwechselt werden. Ein Container erzeugt eine einzelne View (*TOcView*) für sein Verbunddokument. Diese View kann Teile enthalten, die aus anderen Anwendungen empfangen wurden. Jeder Teil zeichnet sich durch Aufruf einer entfernten View von seinem Server selbst. Container erzeugen *TOcView*-Objekte, und Server erzeugen *TOcRemView*-Objekte. (Ein *TOcRemView*-Objekt kann jedoch auch zum Container werden, wenn der Anwender Objekte in andere Objekte einbettet.)

Trotz der ähnlichen Namen sind *TOcDocument*, *TOcView* und *TOcRemView* nicht Teil des Dokument/View-Modells von ObjectWindows. Durch den Aufbau von OLE ist es günstiger, Daten von ihrer grafischen Darstellung zu trennen. Die Begriffe *Dokument* und *View* drücken diese sogar außerhalb von ObjectWindows Trennung aus.

TOcRemView ist ein COM-Objekt und implementiert die Schnittstelle *IUnknown*.

Siehe auch

Klasse TOcDocument, Klasse TOcView

Öffentlicher Konstruktor

Konstruktor

TOcRemView(TOcDocument& doc, TRegList* = 0, IUnknown* outer = 0);
Eine entfernte View ist immer mit einem *TOcDocument*-Objekt verbunden. Das Dokument lädt und speichert die Daten in einem OLE-Objekt, und die entfernte View zeichnet die Daten im Container-Fenster. Dies bedeutet, daß das Dokument immer zuerst erzeugt werden muß.

Außerdem stellt *regList* in beiden Formen eine Registriertabelle des Dokuments dar. Ein Server, der verschiedenen Arten von Objekten erzeugt, benötigt mehrere Dokument-Registriertabellen, und zwar eine für jede Objektart. Der Parameter *regList* legt den Objekttyp fest, den die View darstellt. *outer* zeigt auf die Schnittstelle *IUnknown* eines übergeordneten Objekts, unter dem sich das neue Objekt selbst aufbauen soll.

Registriertabellen werden mit den Makros BEGIN_REGISTRATION und END_REGISTRATION erstellt.

Der Destruktor für *TOcRemView* ist privat. ObjectComponents gibt das Objekt frei, sobald es nicht mehr benötigt wird.

Siehe auch

Makro BEGIN_REGISTRATION, Klasse TAutoObject, Klasse TOcDocument

Öffentliche Elementfunktionen

Copy

virtual bool Copy();
Kopiert das Objekt in die Zwischenablage. Gibt bei Erfolg **true** zurück.

EvClose

virtual void EvClose();
Das entfernte View-Fenster der Anwendung ruft diese Funktion auf, bevor es sich schließt. *EvClose* löst die Verbindung der View zu allen Teilen, die in ihr dargestellt werden.

GetContainerTitle

virtual LPCOLESTR GetContainerTitle();
Fragt den Namen des Containers ab. Der Server enthält diesen String während der direkten Bearbeitung im Server-Dokument normalerweise in seiner eigenen Titelleiste (wenn der Anwender ein verknüpftes oder eingebettetes Objekt im eigenen Fenster des Servers und nicht des Containers editiert).

GetInitialRect

void GetInitialRect();
Fragt die ursprüngliche Größe und Position des Bereichs ab, in den der Server sein Objekt zeichnen kann. Die Funktion initialisiert *Extent*, ein geschütztes Datenelement, das *TOcRemView* von *TOcView* erbt.

Siehe auch TOcView::Extent

Invalidate

void Invalidate(TOcInvalidate invalid);
Informiert die aktive View des Containers darüber, daß der Server entweder die Inhalte oder das Erscheinungsbild des Objekts geändert hat. Der Parameter *invalid* gibt an, was geändert werden muß. Dies können *invData, invView* oder beide in Kombination mit dem Operator OR (|) sein. Wenn der Container eine Anwendung unter ObjectComponents ist, erzeugt seine aktive View eine OC_VIEWPARTINVALID-Botschaft.

Siehe auch OC_VIEWxxxx-Botschaften, Aufzählungstyp TOcInvalidate

IsOpenEditing

bool IsOpenEditing() const;
Gibt **true** zurück, wenn die View gegenwärtig mit einer offenen Editier-Sitzung beschäftigt ist. Man spricht von offenem Editieren, wenn der Anwender die Aktionsart Öffnen (Open) eines Objekts wählt. Offenes Editieren findet im eigenen Rahmenfenster des Servers statt, anders als direktes Bearbeiten im Container-Dokument, das im Fenster des Containers ausgeführt wird. Entfernte View-Objekte werden für beide Arten des Editierens verwendet.

Load

bool Load(IStorage* storageI);
Liest aus *storageI* Informationen, die für die entfernte View spezifisch sind. Diese Informationen sind Bestandteil der Daten, die der Server in der Datei des Containers speichert, wenn er zum Speichern eines Objekts aufgefordert wird. *Load* gibt bei Erfolg **true** zurück.

IStoarage ist ein Zeiger auf eine OLE-Schnittstelle. *storageI* kann ein Zeiger auf ein *TOcStorage*-Objekt sein, das die Implementierung dieser Schnittstelle unter ObjectComponents darstellt.

Siehe auch

TOcRemView::Save

Rename

virtual void Rename();
Aktualisiert den Namens-String, den ObjectComponents erzeugt, um die Teile in einem Verbunddokument zu unterscheiden.*TOcRemView* ruft *Rename* während der Erzeugung auf, um herauszufinden, wie der Container das Objekt nennen will. Normalerweise muß *Rename* nicht direkt angesprochen werden.

Save

bool Save(IStorage* storageI);
Schreibt in *storageI* Informationen, die für die entfernte View spezifisch sind. Diese Informationen sind Bestandteil der Objektdaten, die in der Verbunddatei des Containers gespeichert werden. Gibt bei Erfolg **true** zurück.

IStoarage ist ein Zeiger auf eine OLE-Schnittstelle. *storageI* kann ein Zeiger auf ein *TOcStorage*-Objekt sein, das die Implementierung dieser Schnittstelle unter ObjectComponents darstellt.

Siehe auch TOcRemView::Load

Struktur TOcSaveLoad ocf/ocview.h

Enthält Informationen, die eine View beim Laden und Speichern von OLE-Objektteilen verwendet. Die Botschaften OC_VIEWLOADPART und OC_VIEWSAVEPART enthalten in ihren *IParam*s einen Zeiger auf diese Struktur.

Die *TOleView*-Prozesse laden und speichern Botschaften. Wenn Sie mit dem Dok/View-Modell von ObjectWindows programmieren, müssen Sie die Struktur *TOcSaveLoad* nicht direkt ansprechen. Beispiele für die Verarbeitung der Botschaften zum Laden oder Speichern finden sich im Quellcode für die Methoden *EvViewSavePart* und *EvViewLoadPart* in *TOleView*.

Siehe auch OC_VIEWxxxx-Botschaften, TOleView::EvViewSavePart, TOleView::EvViewLoadPart

Öffentliche Elementfunktionen

Release

bool Release;
Ist **true**, wenn die View das Speicherabteil-Objekt für zukünftige Dateioperationen behalten soll und **false**, wenn sie das Speicherabteil-Objekt nach einmaligem Gebrauch nicht mehr benötigt.

Storagel

IStorage far* Storagel;
Zeigt auf das Speicher-Objekt, das den Teil enthält. ObjectComponents implementiert die Standard-OLE-Schnittstelle *IStorage* in *TOcStorage*. Daher kann *TOcStorage* verwendet werden, um *IStorage* zu erzeugen.

Klasse TOcScaleFactor ocf/ocview.h

Die Klasse *TOcScaleFactor* überträgt die Informationen vom Container zum Server, wie der Container sein Dokument skalieren möchte. Wenn beispielsweise der Container einen Zoom-Befehl besitzt und der Anwender das Dokument um 120% vergrößern will, sollte der Server den Skalierungsfaktor beim Zeichnen von in den Container eingebetteten Objekten anpassen.

ObjectComponents übergibt eine Referenz auf ein *TOcScaleFactor*-Objekt im *lParam* der OC_VIEWGETSCALE- und OC_VIEWSETSCALE-Botschaften. Wenn ein Container OC_VIEWSETSCALE-Informationen empfängt, liest er die Skalierungswerte aus und kann sie dann in seiner Zeichenprozedur verwenden.

TOcScaleFactor speichert Informationen zur Skalierung in seinen zwei *TSize*-Elementen *SiteSize* und *PartSize*. Die Namen beziehen sich auf den Bereich, in dem der Container ein Objekt zeichnen möchte (die Lage) und auf das Objekt selbst (den Teil). Die Werte der Elemente müssen jedoch nicht die tatsächliche Größe des Ortes oder des Teils angeben. Entscheidend ist das Verhältnis der zwei Größen. Wenn die Werte von *SiteSize* doppelt so groß wie die Werte von *PartSize* sind, wird der Server aufgefordert, das Objekt in seiner doppelten Standardgröße zu zeichnen.

Wenn man mit ObjectWindows programmiert, übernimmt die Klasse *TOleWindow* die Skalierung. Beispiele für die Behandlung von Skalierungsaufgaben ohne ObjectWindows finden sich im Quellcode der folgenden *TOleWindow*-Methoden: *EvGetViewScale, EvViewSetScale* und *SetupDC*.

Siehe auch

OC_VIEWxxxx-Botschaften, TOleWindow::EvViewGetScale, TOleWindow::EvViewSetScale, TOleWindow::SetupDC, Klasse TSize

Öffentliche Konstruktoren

Konstruktoren

Form 1

TOcScaleFactor();

Initialisiert die Maße des Orts und des Teils auf 1, so daß der Skalierungsfaktor 100% beträgt.

Form 2

TOcScaleFactor(const RECT& siteRect, const TSize& partSize);

Baut den ursprünglichen Skalierungsfaktor auf den Werten der vorgegebenen Rechteckstruktur auf Berechnet die Ausmaße des Rechtecks *siteRect* und setzt sie in *SiteSize*. Kopiert *partSize* nach *PartSize*.

Form 3

TOcScaleFactor(const BOleScaleFActor far& scaleFactor);

Baut den ursprünglichen Skalierungsfaktor auf den Werten in *scaleFactor* auf. *BOleScaleFactor* ist eine Struktur, die die Support-Bibliothek BOCOLE intern für Skalierungsinformationen verwendet. Die Struktur muß normalerweise nicht direkt angesprochen werden.

Normalerweise muß ein *TOcScaleFactor*-Objekt nicht direkt erzeugt werden. ObjectComponents erzeugt dieses Objekt und übergibt es in der OC_VIEWGETSCALE- oder OC_VIEWSETSCALE-Botschaft.

Destruktor

~TOcScaleFactor();

Siehe auch TOcScaleFactor::PartSize, TOcScaleFactor::SiteSize, Klasse TSize

Öffentliche Datenelemente

PartSize

TSize PartSize;

Enthält zwei Werte, die die horizontalen und vertikalen Standardausmaße eines Server-Objekts beschreiben. Die Werte in *PartSize* müssen keine tatsächlichen Maße sein. Entscheidend ist das Verhältnis der hier gegebenen Werte zu den Werten in *SiteSize*. Dieses Verhältnis legt fest, wie ein Bild skaliert wird.

Siehe auch TOcScaleFactor::SiteSize, Klasse TSize

SiteSize

TSize SiteSize;

Enthält zwei Werte, die die horizontalen und vertikalen Ausmaße des Bereichs beschreiben, den ein Container zur Anzeige eines verknüpften oder eingebetteten Objekts vorgesehen hat. Die Werte in *SiteSize* müssen keine tatsächlichen Maße sein. Entscheidend ist das Verhältnis der hier gegebenen Werte zu den Werten in *PartSize*. Dieses Verhältnis legt fest, wie ein Bild skaliert wird.

Siehe auch TOcScaleFactor::PartSize, Klasse TSize

Öffentliche Elementfunktionen

operator =

Form 1

TOcScaleFactor& operator = (const BOleScaleFactor far& scaleFactor);

Kopiert die Werte in eine *BOleScaleFactor*-Struktur. Die Support-Bibliothek BOCOLE verwendet diese Struktur intern für Skalierungsinformationen.

Form 2

TOcScaleFactor& operator = (const TOcScaleFactor& scaleFactor);

Kopiert einen *TOcScaleFactor* in einen anderen.

Beide Formen des Zuweisungsoperators kopieren die Werte von einem Skalierungsobjekt in ein anderes.

Siehe auch OC_VIEWxxxx-Botschaften

GetScale

uint16 GetScale();
Ermittelt einen Prozentwert, der das Verhältnis der Größe des Teils zu der Größe des Ortes ausdrückt. Beispielsweise gibt *GetScale* 200 zurück, wenn die Teilgröße 20 x 20 und die Größe des Ortes 40 x 40 beträgt.

Siehe auch TOcScaleFactor::SetScale

GetScaleFactor

void GetScaleFactor(BOleScaleFactor far& scaleFactor) const;
Setzt die Werte aus dem *TOcScaleFactor*-Objekt in *scaleFactor* ein. *BOleScaleFactor* ist eine Struktur, die die Support-Bibliothek BOCOLE intern für Skalierungsinformationen verwendet. Die Struktur muß normalerweise nicht direkt angesprochen werden.

IsZoomed

bool IsZoomed;
Gibt **true** zurück, wen die für den Teil und den Ort gespeicherten Größen nicht zusammenpassen.

SetScale

void SetScale(uint16 percent);
Legt das Verhältnis der Größe des Teils zur Größe des Ortes fest. Genauer gesagt, setzt *SetScale* die Größe des Teils auf 100 und die Größe des Ortes auf *percent*.

Siehe auch TOcScaleFactor::GetScale

Aufzählungstyp TOcScrollDir ocf/ocobject.h

enum TOcScrollDir
Das Ereignis OC_VIEWSCROLL informiert den Container, wenn der Anwender eine Ziehbewegung ausführt, die im Fenster einen Bildlauf durchführt. Der *lParam* der Botschaft WM_OCEVENT enthält einen dieser Werte, um anzuzeigen, in welche Richtung das Fenster einen Bildlauf ausführen soll.

Konstante	Bedeutung
sdScrollUp	Im Dokument nach oben rollen.
sdScrollDown	Im Dokument nach unten rollen.
sdScrollLeft	Im Dokument zum linken Rand rollen.
sdScrollRight	Im Dokument zum rechten Rand rollen.

Siehe auch

Ereignisbehandlungsroutine EvOcViewScroll, OC_VIEWxxxx-Botschaften, TOleView::EvOcViewScroll, TOleWindow::EvOcViewScroll, Botschaft WM_OCEVENT

Struktur TOctool barInfo ocf/ocview.h

Die Botschaft OC_VIEWSHOWTOOLS enthält in ihrem *lParam* einen Zeiger auf diese Struktur. Die Botschaft fragt beim Server Handle für ihre Mauspaletten an, damit der Containers sie in seinem eigenen Fenster darstellen kann. Dies geschieht beim direkten Bearbeiten im Container-Dokument, wenn der Anwender ein Objekt im Container öffnet, um es zu editieren.

Die Struktur besitzt vier Felder. Daher kann der Server für bis zu vier Mauspaletten Handle zurückgeben. Jede Mauspalette befindet sich an einer anderen Stelle des Client-Bereichs des Containers.

Beispiele hierfür finden sich im Quellcode für *TOleWindow::EvOcViewShowTools* und *TOleView::EvOcVieShowTools*. Die Standard-Implementierung dieser Methoden sieht eine Mauspalette im oberen Teil des Client-Bereichs vor. Die Botschaft OV_VIEWSHOWTOOLS muß direkt bearbeitet werden, um für den Container mehrere Mauspaletten vorzusehen.

Siehe auch

TOleView::EvOcViewShowTools, TOleWindow::EvOcViewShowTools

Öffentliche Elementfunktionen

HBottomTB

HWND HBottom TB;

Enthält ein Handle auf die Mauspalette, die der Server in den unteren Teil des Client-Bereichs des Containers setzen möchte.

HFrame

HWND HFrame;

Wenn *show* **true** ist, wenn der Server aufgefordert wird, seine Mauspalette darzustellen, enthält *HFrame* ein Handle auf das Rahmenfenster, in dem

die Mauspalette erscheinen soll. Wenn *Show* **false** ist, enthält *HFrame* ein Handle auf das eigene Rahmenfenster des Servers.

Siehe auch

TOctool barInfo::Show

HLeftTB

HWND HLeftTB;
Enthält ein Handle auf die Mauspalette, die der Server an den linken Rand des Client-Bereichs des Containers setzen möchte.

HRightTB

HWND HRightTB;
Enthält ein Handle auf die Mauspalette, die der Server an den rechten Rand des Client-Bereichs des Containers setzen möchte.

HTopTB

HWND HTopTB;
Enthält ein Handle auf die Mauspalette, die der Server in den oberen Teil des Client-Bereichs des Containers setzen möchte.

Show

bool Show;
Ist **true**, wenn der Server seine Mauspalette darstellen soll oder **false**, wenn er sie verbergen soll.

Klasse TOcVerb ocf/ocpart.h

Enthält Informationen zu einer einzelnen Aktionsart, die ein Server für seine Objekte unterstützt.

Eine Aktionsart ist eine Aktion, die der Server mit einem seiner Objekte durchführen kann. Ein Server, der Textobjekte erzeugt, könnte beispielsweise die Aktionsart Bearbeiten unterstützen. Ein Server für Klangobjekte kann Bearbeiten, Wiedergabe und Rückspulen unterstützen.

Wählt der Anwender ein Objekt in einer Verbunddatei aus, fragt der Container beim Objekt *TOcPart* eine Liste der Aktionsarten an, die er ausführen kann. Der Container zeigt die Aktionsarten in seinem Menü Bearbeiten an. Der Befehl zum Aufzählen von Aktionsarten lautet *TOcPart::EnumVerbs*.

Bei jedem Auswählen eines Teils durch den Anwender paßt der Container sein Menü Bearbeiten durch das Hinzufügen eines Eintrags zur Bearbeitung des Objekts an. Ist das Objekt beispielsweise Teil eines Quattro Pro-Arbeitsblattes, fügt der Container seinem Menü Bearbeiten den Befehl Arbeitsblatt Objekt hinzu. Wählt der Anwender diesen Befehl aus, zeigt der Container ein Popup-Menü mit den Arbeitsblatt-Aktionsarten Bearbeiten und Öffnen.

Der Quellcode für *TOleWindow::CeEdirObject* in OLEWINDO.CPP enthält ein Beispiel für die Implementierung dieser Einträge im Menü Bearbeiten.

Siehe auch TOcPart::EnumVerbs

Öffentlicher Konstruktor

Konstruktor

TOcVerb();
Erzeugt ein leeres Aktionsart-Objekt.

Öffentliche Datenelemente

CanDirty

bool CanDirty;
Ist **true**, wenn die Ausführung der Aktionsart das Objekt so verändern kann, daß es danach eventuell gespeichert oder neu gezeichnet werden muß. Beispielsweise ist das Feld *CanDirty* einer Aktionsart Bearbeiten immer **true** und das Feld *CanDirty* einer Aktionsart Wiedergabe normalerweise **false**.

TypeName

LPCOLESTR TypeName;
Zeigt auf den Namen des Objekttyps, zu dem diese Aktionsart gehört. Der Container zeigt diesen Namen normalerweise im Eintrag Objekt seines Menüs Bearbeiten an. Hat der Anwender beispielsweise ein vom Server eingefügtes Objekt beim Ausführen von Schritt 15 der ObjectWindows-Einführung ausgewählt, ist *TypeName* "Drawing Pad", und das Menü Bearbeiten des Containers sollte einen Eintrag haben, der "Drawing Pad" lautet. Die Auswahl dieses Eintrags ruft ein Popup-Menü auf, das alle Aktionsarten des Bildes enthält.

Der String *TypeName* stammt aus dem Wert, den der Server für den Schlüssel *menuname* in seiner Registriertabelle des Dokuments registriert hat.

Siehe auch TOcPart::EnumVerbs, Registrierschlüssel menuname

VerbIndex

uint VerbIndex;
Enthält die Indexnummer, die diese Aktionsart in der Liste der möglichen Aktionsarten des Servers identifiziert. Die erste Aktionsart ist immer 0 und wird als Standard-Aktionsart betrachtet. Führt der Anwender einen Doppelklick auf das Objekt aus, sollte der Container den Server auffordern, seine Standard-Aktionsart auszuführen.

VerbName

LPCOLESTR VerbName;
Zeigt auf den Namen der Aktionsart. Diesen String fügt der Container seinem Menü Bearbeiten hinzu.

Klasse TOcView — ocf/ocview.h

Basisklasse

TUnknown
TOcView verwaltet die Darstellung des Verbunddokuments eines Containers, das verknüpfte oder eingebettete Objekte enthält. Jedes Objekt im Dokument wird von einem Objekt des Typs *TOcPart* dargestellt. Die View des Dokuments weiß, welche Teile ausgewählt oder aktiviert wurden. Sie führt die Bildläufe des Fensters durch und merkt sich, welche Teile sichtbar sind. Sie überträgt mittels der Zwischenablage oder Ziehen-und-Ablegen-Operationen Teile in das Dokument oder aus dem Dokument.

Jede *TOcView* besitzt ein entsprechendes *TOcDocument*. Das Dokument-Objekt von ObjectComponents implementiert die OLE-Schnittstellen, die die Daten in einem Verbunddokument manipulieren. *TOcView* implementiert die Schnittstelle, die das Erscheinungsbild eines Verbunddokuments beeinflußt.

TOcView ist ein COM-Objekt und implementiert die Schnittstelle *IUnknown*.

Siehe auch — Klasse TOcApp, Klasse TOcDocument, Klasse TOcPart, Klasse TUnknown

Öffentlicher Konstruktor

Konstruktor

TOcView(TOcDocument& doc, TRegList* regList = 0, IUnknown* outer=0);
doc bezieht sich auf das Objekt *TOcDocument*, das der View entspricht. *TOcDocument* verwaltet die Daten in einem Verbunddokument, und

TOcView verwaltet das Erscheinungsbild eines Dokuments auf dem Bildschirm.

regList ist die Registrierstruktur für ein bestimmtes Dokument. Mit den Makros BEGIN_REGISTRATION und END_REGISTRATION kann ein Objekt vom Typ *TRegList* erzeugt werden.

outer ist die Stammschnittstelle eines umgebenden Objekts, in dem die neue View sich einpaßt.

Siehe auch

Makro BEGIN_REGISTRATION, Klasse TAutoObject, Klasse TOcDocument

Öffentliche Elementfunktionen

ActivatePart

bool ActivatePart(TOcPart* part);
Versucht, den angegebenen Teil zu aktivieren (durch Aufruf von *TOcPart::Activate*). Gibt **true** zurück, wenn der betreffendene Teil aktiv wird. Andernfalls wird **false** zurückgegeben. War bereits ein anderer Teil aktiv, wird dieser zuerst deaktiviert.

Siehe auch

TOcPart::Activate, TOcView::ActivePart, TOcView::GetActivePart

BrowseClipboard

bool BrowseClipboard(TOcInitInfo& initInfo);
Stellt das Dialogfeld Inhalte einfügen dar, das die verfügbaren Formate für die Daten, die sich gegenwärtig in der Zwischenablage befinden, anzeigt. Der Anwender kann auswählen, welches Format er einsetzen will. Gibt **true** zurück, wenn der Anwender Daten einsetzt und **false**, wenn der Anwender die Aktion abbricht.

Erzeugen Sie zuerst *initInfo* durch Übergabe der View an den Konstruktor *TOcInitInfo*. *BrowseClipboard* stattet *initInfo* mit Informationen über das Objekt aus. Verwenden Sie dann *initInfo*, um einen neuen *TOcPart* zu erzeugen.

Diese Funktion ruft *TOcApp::BrowseClipboard* auf.

Siehe auch

TOcApp::BrowseClipboard, TOcApp::BrowseLinks, Klasse TOcInitInfo, Klasse TOcPart

BrowseLinks

bool BrowseLinks();
Zeigt das Dialogfeld Verknüpfungen an, das alle verknüpften Objekte im Verbunddokument angibt und womit sie verknüpft sind. Der Anwender

kann die angezeigten Verknüpfungen verändern, beispielsweise, um eine Datei, die verschoben wurde, erneut anzubinden. Gibt **false** zurück, wenn das Dialogfeld durch einen Fehler nicht dargestellt wurde oder wenn der Anwender das Dialogfeld abbricht.

Siehe auch TOcApp::BrowseClipboard

Copy

bool Copy(TOcPart* part);
Erzeugt eine Kopie eines verknüpften oder eingebetteten Objekts und setzt diese Kopie in die Zwischenablage. Gibt **true** zurück, wenn die Operation erfolgreich war. Rufen Sie *Copy* als Antwort auf die Befehle Ausschneiden oder Kopieren aus dem Menü Bearbeiten auf.

EvActivate

void EvActivate(bool activate);
Ein Container ruft diese Funktion auf, wenn eines seiner Fenster den Fokus erhält, während eines seiner verknüpften oder eingebetteten Objekte direkt im Container-Dokument bearbeitet wird. *EvActivate* stellt den Fokus für die im Server-Dokument aktivierte View wieder her. Wenn der Anwender beispielsweise im Client-Fenster einen MDI-Rahmen anklickt, muß das Client-Fenster den Fokus zurück auf die View geben. Diese stellt ihrerseits den Fokus für den aktivierten Teil wieder her. Ein Teil, der direkt im Container-Dokument bearbeitet wird, sollte den Fokus immer behalten.

activate sollte **true** sein, wenn das Fenster den Fokus erhält und **false**, wenn es ihn verliert.

Siehe auch TOcView::EvClose, TOcView::EvResize, TOcView::EvSetFocus

EvClose

virtual void EvClose();
Ein Container ruft diese Funktion auf, um ObjectComponents darüber zu informieren, daß das zur View gehörige Fenster geschlossen wurde.

Siehe auch TOcView::EvActivate, TOcView::EvResize, TOcView::EvSetFocus

EvResize

void EvResize();
Ein Container ruft diese Funktion auf, um OLE darüber zu informieren, daß das zur View gehörige Fenster seine Größe ändert. OLE benötigt die-

se Information eventuell, um einem Server zu ermöglichen, seine Mauspalette während der direkten Bearbeitung im Container-Dokument anzupassen.

Siehe auch TOcView::EvActivate, TOcView::EvClose, TOcView::EvSetFocus

EvSetFocus

bool EvSetFocus(boolset);
Ein Container ruft diese Funktion auf, um OLE darüber zu informieren, daß das zur View gehörige Fenster den Eingabefokus entweder erhalten oder verloren hat. *set* sollte **true** sein, wenn das Fenster den Fokus erhalten hat und **false**, wenn es den Fokus verloren hat.

Die Funktion gibt **false** zurück, wenn die View den Fokus nicht erhalten kann. Dies ist der Fall, wenn ein Objekt der View direkt im Container-Dokument bearbeitet wird. Solche Objekte behalten den Fokus, bis die Bearbeitungssitzung beendet wird.

Siehe auch TOcView::EvActivate, TOcView::EvResize, TOcView::EvSetFocus

GetActivePart

TOcPart* GetActivePart();
Gibt den gegenwärtig aktiven Teil zurück. Enthält die View keinen aktiven Teil, ist der Rückgabewert 0.

Siehe auch TOcView::ActivePart, TOcView::activatePart

GetOcDocument

TOcDocument& GetOcDocument();
Gibt das ObjectComponents-Dokumentzurück, das zu der View gehört. Views und Dokumente arbeiten paarweise zusammen. *TOcView* verwaltet das Erscheinungsbild eines Verbunddokuments, und *TOcDocument* verwaltet die Daten in ihm.

Siehe auch Klasse TOcDocument,TOcView::OcDocument

GetOrigin

TPoint GetOrigin() const;
Gibt die physikalischen Koordinaten zurück, die gerade in der linken oberen Ecke des Client-Bereichs des Containers abgebildet werden. Programmierer, die mit ObjectWindows arbeiten, können diese Methode ignorieren, da *TOleWindow* die Bildläufe für sie durchführt.

Siehe auch TOcView::Origin, TOcView::ScrollWindow, Klasse TPoint

GetWindowRect

TRect GetWindowRect() const;
Gibt das Rechteck des Clients für das View-Fenster zurück.

InvalidatePart

void InvalidatePart(const TOcPart* part);
Setzt eine OC_VIEWPARTINVALID-Botschaft an das Container-Fenster ab. Antwortet das Container-Fenster mit **false** und zeigt damit an, daß es die Botschaft nicht verarbeitet hat, teilt *InvalidatePart* dem System mit, daß der Bereich innerhalb des Begrenzungsrechtecks des Teils ungültig ist und neu gezeichnet werden muß.

Paste

bool Paste(bool linking = false);
Fügt ein Objekt aus der Zwischenablage in das Verbunddokument ein. Wenn *linking* **true** ist, versucht *Paste*, das neue Objekt einzubetten anstatt es zu verknüpfen. Setzen Sie *linking* auf **true**, wenn Sie den Befehl Verknüpfung einfügen verarbeiten.

RegisterClipFormats

bool RegisterClipFormats(TRegList& regList);
Teilt OLE mit, welche Zwischenablage-Formate das Dokument versteht. Die Formatliste stammt aus *regList*, der Registrierstruktur des Dokuments. Verwenden Sie die Makros BEGIN_REGISTRATION und END_REGISTRATION, um *regList* zu erzeugen. Das Makro REGFORMAT fügt ebenfalls Formateinträge der Zwischenablage in die Struktur ein. Um benutzerdefinierte Zwischenablage-Formate zu registrieren, muß außerdem *TOcApp::AddUserFormatName* aufgerufen werden.

RegisterClipFormats wird bei der Erzeugung der View automatisch aufgerufen.

Siehe auch Makro BEGIN_REGISTRATION, Makro REGFORMAT, TOcApp::AddUserFormatName, TOcView::FormatList

ReleaseObject

virtual void ReleaseObject();
Rufen Sie diese Funktion anstatt zu löschen zum Entfernen eines nicht mehr benötigten *TOcView*-Objekts auf. *ReleaseObject* dekrementiert den interenen Referenzzähler der View und trennt die View von ihrem Fenster.

Siehe auch TOcView::SetupWindow

Rename

virtual void Rename();
Informiert OLE, wenn der einem Verbunddokument zugewiesenen Name sich geändert hat. OLE aktualisiert seine internen Daten. Außerdem übergibt das zugehörige *TOcDocument*-Objekt den neuen Namen an jedes verknüpfte oder eingebettete Objekt, das es enthält.

ScrollWindow

void ScrollWindow(int dx, int dy);
Macht neue Bereiche des Dokuments durch die Anpassung des Ursprungs des Container-Fensters sichtbar. *dx* und *dy* sind der horizontale und vertikale Versatz, der dem Ursprung hinzugefügt wird. Die Funktion wird üblicherweise als Antwort auf Botschaften der Fenster-Rollbalken oder der Pfeiltasten aufgerufen.

Siehe auch TOcView::GetOrigin, TOcView::Origin

SetLink

void SetLink(bool pasteLink);
Setzt ein internes Flag ,das bestimmt, ob Operationen zum Einfügen verknüpfte oder eingebettete Objekte erzeugen. Genauer gesagt, verändert *SetLink* die Priorität der für das Dokument registrierten Zwischenablage-Formate. Die ursprünglichen Prioritäten werden durch den ersten Parameter des Makros REGFORMAT gesetzt. Wenn *pasteLink* **true** ist, verschiebt *SetLink* das Quellformat der Verknüpfung an den Anfang der Liste verschoben. Wenn *pasteLink* **false** ist, wird das Quellformat der Verknüpfung an seiner ursprünglichen Position hinter der Einbettungsquelle wiederhergestellt.

Normalerweise muß *SetLink* nicht direkt angesprochen werden, da die Methode *Paste* dies übernimmt.

Siehe auch Makro REGFORMAT, TOcView::Paste

SetupWindow

void SetupWindow(HWND hWin);
Informiert die View, welches Fenster zu ihr gehört. Die View schickt manchmal Benachrichtigungsbotschaften an ihr Fenster. Normalerweise sollte diese Funktion vom Element *SetupWindow* der Fensterklasse des Containers aufgerufen werden. *TOleWindow* führt diese Aufgabe automatisch aus.

Siehe auch

OC_VIEWxxxx-Botschaften, TOleWindow::SetupWindow, TOcView::Win

Geschützter Destruktor

Geschützter Destruktor

~TOcView();
Entfernt das View-Objekt.

Geschützte Elementfunktionen

ForwardEvent

Form 1
uint ForwardEvent(int eventId, const void far* param);
Form 2
uint ForwardEvent(int eventId, uint32 param = 0);
Beide Formen senden eine WM_OCEVENT-Botschaft an das Fenster des Containers. Der Parameter *eventID* wird zum *wParam* der Botschaft und sollte eine der Konstanten OC_APPxxxx oder OC_VIEWxxxx sein. Der zweite Parameter wird zum *lParam* der Botschaft. Er kann entweder ein Zeiger (Form 1) oder ein Integerwert (Form 2) sein. Welche Form Sie verwenden, hängt von den Informationen ab, die ein bestimmtes Ereignis benötigt, um seinen *lParam* zu senden.

Siehe auch

OC_APPxxxx-Botschaften, OC_VIEWxxxx-Botschaften, TOcView::Win, WM_OCEVENT-Botschaft

Init

void Init(TRegList* regList);
Initialisiert ein neu erzeugtes View-Objekt. *Init* wird von beiden *TOcView*-Konstruktoren aufgerufen. Normalerweise muß diese Funktion nicht direkt angesprochen werden. *TRegList* ist der Datentyp, der alle Registrierschlüssel und die dazugehörigen Werte für eine einzelne Registriertabelle enthält. *regList* muß eine Dokument-Registriertabelle sein (die Struktur, die von den Registriermakros erzeugt wird und üblicherweise *DOcReg* heißt).

Init macht diese View zur aktiven View des Dokuments, stellt die Verbindung zur Support-Bibliothek BOCOLE her und registriert die unterstützten Zwischenablage-Formate.

Shutdown

void Shutdown();
Wird vom Destruktor abgeleiteter Klassen aufgerufen, um Hilfsobjekte, die die View intern enthält, freizugeben.

Siehe auch Öffentlicher Konstruktor und Destruktor TOcView

Geschützte Datenelemente

ActivePart

TOcPart* GetActivePart();
Gibt den gegenwärtig aktiven Teil zurück. Enthält die View keinen aktiven Teil, ist der Rückgabewert 0.

Siehe auch TOcView::ActivePart, TOcVoew::ActivatePart

Extent

TSize Extent;
Enthält die gegenwärtige Breite und Höhe des Client-Bereichs des Container-Fensters. Beide werden in Geräteeinheiten gemessen.

Siehe auch TOcView::GetWindowRect

FormatList

TOcFormatList FormatList;
Enthält Informationen zu allen Zwischenablage-Formaten, die das Verbunddokument unterstützt. Die Liste wird aus Informationen erstellt, die die Anwendung für die Dokumenttypen, die sie unterstützt, registriert.

Siehe auch Klasse TOcFormatList, TOcView::RegisterClipFormats

Link

int Link;
Wird intern von der Methode *Paste* verwendet, um die Priorität des Verknüpfungsquellen-Formats einzustellen.

OcApp

TOcApp& OcApp;
Eine View speichert die Anwendung, der sie gehört, in diesem geschützten Datenelement.

OcDocument

TOcDocument& OcDocument;
Eine View speichert das Dokument-Objekt, dem die View gehört, in diesem geschützten Datenelement. Das View-Objekt verwaltet das Erscheinungsbild eines Verbunddokuments, und das Dokument-Objekt verwaltet die Daten.

Siehe auch TOcView::GetOcDocument

Origin

TPoint Origin;
Enthält die Koordinaten des Punktes, der gerade in der oberen, linken Ecke des Client-Bereichs des Container-Fensters abgebildet wird.

Siehe auch TOcView::GetOrigin

Win

HWND Win;
Enthält ein Handle auf das Fenster, in dem sich die View selbst zeichnet. Die Methode *ForwardEvent* schickt Botschaften an dieses Fenster.

Siehe auch TOcView::ForwardEvent, TOcView::SetupWindow

WinTitle

string WinTitle;
Enthält den Original-Überschriftsstring des Container-Fensters. Die Überschrift wird normalerweise angepaßt, während sich der Anwender im Dokument von einem Teil zum anderen bewegt. Wenn kein Teil aktiv ist, stellt die View den Fenstertitel mit diesem Original-String wieder her.

Struktur TOcViewPaint ocf/ocview.h

Die Botschaft OC_VIEWPAINT enthält in ihrem *lParam* einen Zeiger auf diese Struktur. Die Botschaft unterrichtet einen Server, daß er das Zeichnen eines Objekts aktualisieren soll. Die Struktur enthält Informationen darüber, welcher Bereich neu gezeichnet werden muß. Normalerweise sollte ein Programm durch Aufruf von Zeichenmethoden für das Fenster oder die View, die die Botschaft erhalten, antworten. Beispiel finden sich im Quellcode für *TOleWindow::EcOcViewPaint* und *TOleView::EvOcViewPaint*.

Siehe auch OC_VIEWxxxx-Botschaften, TOleView::EvOcViewPaint, TOleWindow::EvOcViewPaint

Öffentliche Datenelemente

Aspect

TOcAspect Aspect;
Enthält einen Aufzählungswert, der angibt, wie der Teil gezeichnet werden muß. Ein einzelnes Objekt kann häufig auf mehrere Arten gezeichnet werden. Beispielsweise kann der Server den gesamten Inhalt eines Objekts anzeigen, eine verkleinerte Darstellung des Inhalts oder ein Symbol, das den Objekttyp anzeigt, ohne seinen individuellen Inhalt anzugeben.

Siehe auch Aufzählungstyp TOcAspect

Clip

TRect* Clip;
Legt den Bereich fest, in den der Teil zeichnen darf. Der Server kann die Ausgabe an diesen Bereich beschneiden, damit nicht außerhalb des zugewiesenen Bereichs gezeichnet werden kann.

Siehe auch Klasse TRect

DC

HDC DC;
Enthält ein Handle auf den Gerätekontext, in dem das Neuzeichnen stattfinden soll.

Part

TOcPart* Part;
Zeigt auf den Teil, der neu gezeichnet werden muß. Dieses Element kann zur Aufforderung des Teils, sich neu zu zeichnen, verwendet werden. In der aktuellen Implementierung von ObjectComponets wird dieses Element nicht verwendet.

Siehe auch Klasse TOcPart

Pos

TRect* Pos;
Gibt die linke, obere Ecke des Server-Objekts, das ungültig geworden ist und neu gezeichnet werden muß, an.

Siehe auch Klasse TRect

Klasse TOleAllocator ocf/oleutil.h

Eine .EXE-Anwendung zum Verknüpfen und Einbetten erzeugt ein Speicherzuweisungs-Objekt, um OLE mitzuteilen, welche Speicherverwaltung das System benutzen sollte, wenn es Speicher für den Server zuweist oder freigibt. Sofern nicht besondere Anforderungen an die Speicherverwaltung bestehen, ist es am einfachsten, OLE den Speicher standardmäßig zuweisen zu lassen.

Normalerweise müssen Sie beim Programmieren einer Anwendung, die Verknüpfung und Einbettung verwendet, das Speicherzuweisungs-Objekt nicht direkt erzeugen, weil dies das *TOcApp*-Objekt für Sie erledigt. Die einzigen Anwendungen, die Speicherzuweisungs-Objekte direkt erzeugen, sind Automatisierungs-Server, die keine Verknüpfung und Einbettung unterstützen. Da Automatisierungs-Server keine *TOcApp*-Objekte erzeugen, müssen sie *TOleAllocator*s erzeugen.

DLL-Server benötigen keine Speicherzuweisungs-Objekte, da das System jeden Allokator verwendet, den der .EXE-Client bestimmt.

Siehe auch

Klasse TOcApp

Öffentliche Konstruktoren und Destruktor

Konstruktor

Form 1

TOleAllocator(IMalloc* mem = 0);
Initialisiert die OLE-Systembibliothek und registriert einen benutzerspezifischen Speicher-Allokator, wenn *mem* ungleich Null ist. Sofern nicht besondere Anforderungen an die Speicherverwaltung bestehen, ist es am einfachsten, OLE den Speicher standardmäßig zuweisen zu lassen. In der Dokumentation zu OLE finden Sie unter der Schnittstelle *IMalloc* weitere Informationen, wie Sie ihren eigenen Allokator implementieren können.

Form 2

TOleAllocator();
Weist OLE an, das benutzerdefinierte Speicherzuweisungs-Objekt zu verwenden. Initialisiert nicht die OLE-Systembibliothek. In .EXE-Anwendungen initialisiert das Registrar-Objekt die OLE-Bibliothek. Für DLL-Server stellt der .EXE-Client den Allokator zur Verfügung.

Destruktor

~TOleAllocator();
Gibt das Speicherzuweisungs-Objekt frei (entweder den Standard-Allokator oder den benutzerdefinierten Allokator) und hebt die Initialisierung des OLE-Systems auf.

Siehe auch Klasse TOcRegistrar, Klasse TRegistrar

Öffentliche Elementfunktionen

Alloc

void far* Alloc(unsigned long size);
Ruft die Methode *Alloc* auf dem aktiven Speicher-Allokator auf, um einen Speicherblock anzufordern. *size* gibt die Größe des Blocks an. Wenn Sie kein benutzerdefiniertes Speicherzuweisungs-Objekt registriert haben, ruft *Alloc* den Standard-Allokator von OLE auf. Mißlingt die Speicheranforderung, gibt *Alloc* 0 zurück.

Siehe auch TOleAllocator::Free_

Free

void Free(void far* block);
Ruft die Methode *Free* auf dem aktiven Speicher-Allokator auf, um einen Speicherblock freizugeben, der zuvor mit *Alloc* alloziiert wurde. *block* zeigt auf die Basis des Bereichs, der freigegeben werden soll. Wenn Sie kein benutzerdefiniertes Speicherzuweisungs-Objekt registriert haben, ruft *Free* den Standard-Allokator von OLE auf.

Siehe auch TOleAllocator::Alloc_

Öffentliches Datenelement

Mem

IMalloc* Mem;
Zeigt auf das aktive Speicherzuweisungs-Objekt. Wenn Sie kein benutzerdefiniertes Speicherzuweisungs-Objekt registriert haben, zeigt *Mem* auf den Standard-Allokator von OLE.

Klasse TRegistrar ocf/ocreg.h

TRegistrar verwaltet alle Registrieraufgaben für eine Anwendung. Er verarbeitet auf OLE bezogenen Schalter auf der Kommandozeile und zeichnet notwendige Informationen über die Anwendung in der Registrier-Datenbank des Systems auf. Ist die Anwendung bereits in der Datenbank registriert, bestätigt der Registrar, daß die registrierten *path*, *progid* und *clsid* noch korrekt sind. Ist dies nicht der Fall, wird die Anwendung neu registriert.

Jede Anwendung unter ObjectComponents muß ein Registrar-Objekt erzeugen. Wenn Ihre Anwendung Automatisierung, nicht jedoch Verknüpfen und Einbetten unterstützt, erzeugen Sie ein *TRegistrar*-Objekt. Zur Unterstützung von Verknüpfen und Einbetten - allein oder in Verbindung mit Automatisierung - muß statt dessen ein *TOcRegistrar* erzeugt werden. *TOcRegistrar* erweitert *TRegistrar* durch Anbindung der Anwendung an die Schnittstellen der Support-Bibliothek BOCOLE, die Verknüpfung und Einbettung unterstützen.

Die Hauptprozedur einer Anwendung führt normalerweise die folgenden Aktionen mit ihrem Registrar aus:

Erzeugen des Registrars, Übergabe an einen Zeiger auf den Factory-Callback der Anwendung.

Aufrufen von *IsOptionSet* auf, um die Optionen zu prüfen, die möglicherweise die Art des Starts der Anwendung beeinflussen (beispielsweise unsichtbar zu bleiben, wenn der Aufruf für eine Einbettung geschieht).

Aufrufen von *Run*, um in die Meldungsschleife des Programms einzutreten.

Siehe auch

Klasse TOcRegistrar

Öffentlicher Konstruktor und Destruktor

Konstruktor

TRegistrar(TRegList& regInfo, TComponentFactory callback, string& cmdLine, HINSTANCE hInst);
regInfo ist die Registrationsstruktur der Anwendung (üblicherweise wir der Name *appReg* vergeben). *callback* ist die Funktion Factory-Callback, die ObjectComponents aufruft, wenn die Anwendung ein Objekt erstellen muß. Ein ObjectWindows-Programm kann die Klasse *TOleFactory* verwenden, um diesen Callback zu implementieren. *cmdLine* enthält den Kommandozeilen-String, der die Anwendung aufgerufen hat. *hInst* ist die Instanz der Anwendung.

DestruktorDestruktor;TRegistrar

virtual ~TRegistrar();
Der Konstruktor verarbeitet auf OLE bezogene Schalter und entfernt sie aus der Kommandozeile. (Sie können durch den Aufruf von *IsOptionSet* herausfinden, welche Schalter gefunden wurden.) Er initialisiert außerdem einige Einstellungen aus der Registriertabelle der Anwendung. Wenn es sich bei der Anwendung um eine DLL handelt, initialisiert der Konstruktor die globale Variable *DllRegistrar*.

Siehe auch

Klasse string, Typdefinition TComponentFactory, Klasse TOleFactory, TRegistrar::IsOptionSet

Öffentliche Elementfunktionen

CanUnload

virtual bool CanUnload();
Gibt **true** zurück, wenn die Anwendung gegenwärtig keine OLE-Clients bedient. Andernfalls wird **false** zurückgegeben.

CreateAutoApp

TUnknown* CreateAutoApp(TObjectDescriptor app, uint32 options, IUnknown* outer = 0);
Erzeugt eine Instanz einer automatisierten Anwendung. Diese Methode wird üblicherweise von der Callback-Funktion *TComponentFactory* der Anwendung aufgerufen.

app ist die automatisierte Primärklasse des Automatisierungs-Servers, die aus dem Template *TAutoObjectDelete*<> abgeleitet wurde.

options enthält die Modus-Flags der Anwendung. Es handelt sich dabei normalerweise um den gleichen Werte, die an die Factory-Callback-Funktion übergeben werden. Die möglichen Werte werden in *TOcAppMode* aufgezählt.

outer zeigt auf die Schnittstelle *IUnknown* einer äußeren Komponente, unter der sich die Anwendung aggregieren soll.

Der Rückgabewert zeigt auf das neue OLE-Anwendungsobjekt.

Siehe auch

TAutoObjectDelete<>Klasse, Typdefinition TComponentFactory, TRegistrar::CreateAutoObject

CreateAutoObject

Form 1

TUnknown* CreateAutoObject(TObjectDescriptor obj, TServerObject& app);
app ist das automatisierte OLE-Anwendungsobjekt.

obj ist das automatisierte C++ Objekt.

Form 2

TUnknown* CreateAutoObject(const void* obj, const typeinfo& objInfo, const void* app, const typeinfo& appInfo);
app und *obj* entsprechen der Beschreibung in Form 1.

objInfo kennzeichnet den Objekttyp in *obj*. *appInfo* kennzeichnet den Objekttyp in *app*. Beide Werte erhält man durch *typeid*.

CreateAutoObject fordert eine automatisierte Anwendung auf, eines ihrer automatisierten Objekte zu instantiieren. Diese Funktion wird normalerweise von der Callback-Funktion *TComponentFactory* der Anwendung aufgerufen. Welche Form Sie wählen, hängt davon ab, welche Informationen Ihnen zur Kennzeichnung des Objekts, das Sie erzeugen wollen, zur Verfügung stehen.

Siehe auch

Typdefinition TComponentFactory, TRegistrar::CreateAutoApp, typeid, Klasse typeinfo

GetFactory

virtual void far* GetFactory(const GUID& clsid, const GUID far& iid);

Gibt einen Zeiger auf die Factory-Schnittstelle zur Erzeugung des Objekttyps, der durch *clsid* gekennzeichnet ist, zurück. *iid* benennt die betreffende Schnittstelle, die empfangen werden soll. Kann der Registrar keine *iid*-Schnittstelle für *clsid*-Objekte finden, gibt er Null zurück.

ObjectComponents ruft das Element *GetFactory* einer DLL immer dann auf, wenn ein neuer Client die DLL lädt. Normalerweise muß *GetFactory* nicht direkt aufgerufen werden.

GetOptions

uint32 GetOptions() const;

Gibt einen 32-Bit Integerwert zurück, der Bitflags enthält, die den Ausführungsmodus der Anwendung angeben. Einige dieser Flags werden als Antwort auf Kommandozeilen-Schalter gesetzt. Andere werden direkt durch ObjectComponents gesetzt. Eine Liste der Modus-Flags findet sich unter dem Aufzählungstyp *TOcAppMode*.

Siehe auch

Aufzählungstyp TOcAppMode, TRegistrar::IsOptionSet, TRegistrar::ProcessCmdLine, TRegistrar::SetOption

IsOptionSet

bool IsOptionSet(uint32 option) const;

Gibt **true** zurück, wenn eine bestimmte Option auf der Kommandozeile der Anwendung als Flag gesetzt wurde. **false** wird zurückgegeben, wenn die Option nicht gesetzt wurde. Die Flags werden durch die Methode *ProcessCmdLine* gesetzt.

Eine Liste der möglichen Werte, die *option* annehmen kann, findet sich unter dem Aufzählungstyp *TOcAppMode*.

Siehe auch

TOcApp::IsOptionSet, Aufzählungstyp TOcAppMode, TRegistrar::GetOptions, TRegistrar::ProcessCmdLine, TRegistrar::SetOption

ProcessCmdLine

void ProcessCmdLine(string& cmdLine);
Lokalisiert auf OLE bezogene Schalter auf der Kommandozeile der Anwendung (oder solche, die von ObjectComponents an einen DLL-Server übergeben wurden). Die Schalter teilen dem Programm mit, ob es unabhängig oder als Server gestartet wurde, ob es sich registrieren oder abmelden (deregistrieren) sollte, ob es eine Typbibliothek anlegen sollte, und informieren es ebenso über andere Ausführungsbedingungen. *ProcessCmdLine* zeichnet die Gegenwart jedes Flags auf, das es findet. Über einen Aufruf von *IsOptionSet* können Sie die Ergebnisse abfragen.

Die Kommandozeile wird für Sie immer bearbeitet, wenn ein Registrar-Objekt konstruiert wird. Normalerweise müssen Sie diese Funktion nicht direkt aufrufen.

cmdLine enthält den Argumentstring, der dem Programm in seiner Kommandozeile übergeben wird. *ProcessCmdLine* entfernt alle auf OLE bezogenen Schalter aus der Kommandozeile. Auf diese Weise können Sie *cmdLine* zu einem späteren Zeitpunkt für jedes Ihrer eigenen Argumente bearbeiten, ohne sich um OLE-Argumente kümmern zu müssen.

Siehe auch: Klasse string, TRegistrar::IsOption Set

ReleaseAutoApp

void ReleaseAutoApp(TObjectDescriptor app);
Diese Methode wird von der Factory-Callback-Funktion einer Anwendung verwendet, wenn die Anwendung die Anbindung an OLE aufgeben muß, bevor sie sich schließen kann. Das Abkoppeln der Anwendung von OLE ist erforderlich, wenn eine automatisierte Anwendung ihr Anwendungsobjekt für ihre Klasse registriert und damit dem Kontroller erlaubt hat, es zu manipulieren.

RegisterAppClass

void RegisterAppClass();
Teilt OLE mit, daß eine automatisierte Anwendung bereit ist, eine Anwendungsinstanz zu erzeugen. Ist ohne Wirkung, wenn es aus einer Anwendung aufgerufen wird, die Automatisierung nicht unterstützt.

Es ist aus Gründen der Bequemlichkeit zu empfehlen, daß jede Object-Components-Anwendung, auch solche, die Automatisierung nicht unterstützen, beim Start *RegisterAppClass* aufruft und *UnregisterAppClass*, wenn sie schließt. Dieses Vorgehen richtet keinen Schaden an, auch wenn es im Einzelfall überflüssig sein mag, stellt aber sicher, daß Sie das Aufnehmen

der Registrierfunktionen nicht vergessen, wenn Sie später Automatisierung hinzufügen.

Siehe auch: TRegistrar::UnregisterAppClass

Run

virtual int Run();
Rufen Sie diese Funktion auf, um Ihr Programm auszuführen. Wenn die Anwendung als .EXE-Datei erzeugt wurde, läßt *Run* die Anwendung in ihre Nachrichtenschleife eintreten. Ist die Anwendung eine DLL, kehrt *Run* zurück, ohne in die Nachrichtenschleife einzutreten. DLL-Server müssen darauf warten, daß OLE ihre Factory-Funktion aufruft, bevor sie ausgeführt werden können. Der Zweck der Funktion *Run* ist, Ihnen den Aufbau Ihrer Anwendungen wahlweise als .EXE oder als DLL zu gestatten, ohne daß Sie Änderungen am Code vornehmen müßten.

In .EXE-Programmen führt *Run* folgende Schritte aus:

Ruft *RegisterAppClass* auf, wenn die Anwendung automatisiert ist.

Ruft die Factory-Funktion auf, um die Anwendung auszuführen. Die Anwendung tritt in ihre Nachrichtenschleife ein.

Ruft die Factory-Funktion auf, um die Anwendung zu schließen.

Stellt sicher, daß das Verbinderobjekt TOcApp der Anwendung korrekt freigegeben wird.

Siehe auch: TOcApp, TRegistrar::RegisterApp, TRegistrar::Shutdown

SetOption

void SetOption(uint32 bit, bool state);
Modifiziert die Ausführungsmodus-Flags der Anwendung. *bit* enthält Bitflags aus dem Aufzählungstyp *TOcAppMode*. Wenn *state* **true** ist, schaltet *SetOption* die Flags ein. Ist *state* **false**, schaltet es die Flags aus. Ein Aufrufen dieser Funktion sollte niemals erforderlich werden, weil ObjectComponents die Modus-Flags immer wartet.

Siehe auch: Aufzählungstyp TOcAppMode, TRegistrar::GetOptions, TRegistrar::IsOptionSet, TRegistrar::ProcessCmdLine

Shutdown

virtual void Shutdown(IUnknown* releasedObj, uint32 options);
Ruft die Factory-Funktion der Anwendung auf und bittet sie, die Anwendung anzuhalten. Stellt sicher, daß das Verbinderobjekt *TOcApp* der An-

wendung korrekt freigegeben wird. Diese Aufgabe wird bei normalem Gang der Ausführung vom Befehl *Run* ausgeführt. Rufen Sie *Shutdown* auf, um die Anwendung direkt zu beenden.

Siehe auch: TOcApp, TRegistrar::Run

UnregisterAppClass

void UnregisterAppClass();
Gibt bekannt, daß die Anwendung nicht länger für OLE-Aktionen zur Verfügung steht.

Siehe auch: TRegistrar::RegisterAppClass

Geschütztes Datenelement

AppDesc

TAppDescriptor& AppDesc;
Enthält den Deskriptor der Anwendung. ObjectComponents setzt intern einen Anwendungsdeskriptor ein, um Informationen über eine Komponente zu verwalten. (DLLs erhalten, ebenso wie.EXE-Dateien, einen eigenen Anwendungsdeskriptor). *TAppDescriptor* ist nicht dokumentiert, weil es intern verwendet wird und Änderungen unterliegt. Die Registrar-Klassen, *TOcRegistrar* und *TRegistrar*, stellen die unterstützten Schnittstellen zum Anwendungsdeskriptor dar. Der Registrar konstruiert den Deskriptor, und die meisten seiner Elementfunktionen rufen für die Ausführung ihrer Aufgaben Deskriptorfunktionen auf.

Normalerweise werden Sie dieses Datenelement nicht direkt ändern müssen.

Geschützter Konstruktor

Konstruktor

TRegistrar(TAppDescriptor& appDesc);
Der geschützte Konstruktor wird nur von der abgeleiteten Klasse *TOcRegistrar* verwendet. *TAppDescriptor* ist eine Klasse, die beide Registrar-Objekte (*TRegistrar* und *TOcRegistrar*) intern einsetzen, um Informationen über eine Anwendung aufzunehmen.

Klasse TUnknown — ocf/oleutil.h

Implementiert die OLE-Standardschnittstelle *IUnknown*. ObjectComponents leitet einige seiner eigenen Klassen aus *TUnknown* ab, daher müssen Sie sie normalerweise nicht direkt verwenden. Sehr fortgeschrittene Anwender können *TUnknown* aber sinnvoll zum Erzeugen eigener, benutzerdefinierter Component Object Model (COM)-Objekte einsetzen.

Die Klasse *TUnknown* bildet die Basis für die Implementation der Objekt-Aggregation in ObjectComponents. Aggregation erlaubt es, voneinander getrennte Komponenten als ein einziges OLE-Objekt zusammenarbeiten zu lassen. Ein einzelnes Hauptobjekt wird zum äußeren Objekt, und sekundäre Objekte verhalten sich, als wären sie ein Teil des äußeren Objekts. Damit dies funktioniert, muß jedes innere Objekt, dessen *IUnknown*-Schnittstelle angesprochen wird, stattdessen die *IUnknown*-Schnittstelle zurückgeben, die dem äußeren Objekt gehört. Wird beim äußeren Objekt eine Schnittstelle angefordert, die dieses nicht unterstützt, leitet es die Anforderung an die Kette der an es angehängten inneren Objekte weiter. Alle Schnittstellen, die von einem beliebigen Objekt der Aggregation unterstützt werden, sind über die Methode *QueryInterface* des äußeren Objekts verfügbar.

Aggregation wird in der Callback-Funktion *TComponentFactory* eingerichtet. Jede Komponente erhält den *IUnknown*-Zeiger auf ihr äußeres Objekt und gibt ihren eigenen *IUnknown*-Zeiger zurück, damit sie in der Kette sekundärer Objekte plaziert werden kann.

Siehe auch: Typdefinition TComponentFactory

Öffentliche Elementfunktionen

Aggregate

IUnknown& Aggregate(TUnknown& inner);
Versammelt (aggregiert) ein neues Objekt unter dem aktuellen Objekt. *inner* zeigt auf die Schnittstelle *IUnknown* des neuen Objekts. Das aktuelle Objekt speichert *inner*, um auf spätere *QueryInterface*-Aufrufe reagieren zu können. Es ruft auch *AddRef* für den inneren Zeiger auf.

Ist **this** bereits Bestandteil einer Aggregation, wird *inner* an das letzte innere Objekt in der Kette weitergegeben.

Aggregate liefert eine Referenz auf die äußere *IUnknown*-Schnittstelle des Objekts zurück. Das neu hinzugefügte Objekt sollte den Rückgabewert ebenfalls als seinen *Outer*-Zeiger verwenden. Um **this** unter einem Objekt zu aggregieren, das kein *TUnknown*-Objekt ist, rufen Sie stattdessen *SetOuter* auf.

Siehe auch: TUnknown::SetOuter, TUnknown::Outer

GetOuter

IUnknown* GetOuter();
Liefert einen Zeiger auf die äußere *IUnknown*-Schnittstelle des Objekts zurück, die dem Hauptobjekt in einer Gruppe aggregierter Objekte gehört.

Siehe auch: TUnknown::SetOuter, TUnknown::Outer

GetRefCount

unsigned long GetRefCount();
Liefert den Referenzzähler des äußeren Objekt zurück. Wenn **this** nicht Teil der Aggregation ist, gibt *GetRefCount* den eigenen Referenzzähler des Objekst zurück.

Der Referenzzähler gibt an, wie viele Clients Zeiger auf das Objekt enthalten. Der Destruktor verhindert ein Entfernen des Objekts, wenn der Referenzzähler nicht 0 ist.

operator IUnknown&()

operator IUnknown&();
Gibt eine Referenz auf die äußere *IUnknown*-Schnittstelle des Objekts zurück. Erhöht nicht den Referenzzähler des Objekts.

operator IUnknown*()

operator IUnknown*();
Gibt einen Zeiger auf die äußere *IUnknown*-Schnittstelle des Objekts zurück. Vermindert zuvor den Referenzzähler des Objekts.

Siehe auch: TUnknown::operator IUnknown&()

SetOuter

IUnknown* SetOuter(IUnknown* outer);
Weist das Objekt an, sich unter dem Objekt *outer* zu aggregieren. **this** gibt immer *outer* zurück, wenn seine *IUnknown*-Schnittstelle angefordert wird. *SetOuter* gibt dagegen die eigene *IUnknown*-Schnittstelle des Objekts an das äußere Objekt zurück. Es ruft *AddRef* nicht vor der Rückgabe des Zeigers auf.

Ist *outer* gleich 0, ignoriert *SetOuter outer*, gibt aber trotzdem die *IUnknown*-Schnittstelle zurück.

SetOuter wird aufgerufen, damit sich das Objekt unter einem unbekannten äußeren Objekt aggregiert. Ist das äußere Objekt ebenfalls ein *TUnknown*-Objekt, rufen Sie stattdessen *Aggregate* auf. *Aggregate* setzt sowohl den inneren als auch den äußeren Zeiger des Objekts.

Siehe auch: TUnknown::Aggregate, TUnknown::GetOuter, TUnknown::Outer, TUnknown:: operator IUnknown*()

Geschützter Konstruktor und Destruktor

Konstruktor

TUnknown();
Erzeugt ein *TUnknown*-Objekt mit einem anfänglichen Referenzzähler von 0. Zu Beginn ist das Objekt nicht mit irgendeinem anderen Objekt aggregiert.

Diese Elemente sind geschützt, weil nur eine abgeleitete Klasse fähig sein sollte, ein *TUnknown*-Objekt zu konstruieren. *TUnknown* ist als Basis für andere Objekte, nicht als eigenständiges Objekt vorgesehen.

Destruktor

virtual ~TUnknown();
Entfernt das Objekt.

Siehe auch: TUnknown::Aggregate

Geschützte Elementfunktionen

QueryObject

virtual HRESULT QueryObject(const GUID far& iid, void far* far* pif);
Fragt an, ob das Objekt die in *iid* identifizierte Schnittstelle unterstützt. Wenn das Objekt die Schnittstelle unterstützt, gibt die Funktion HR_NOERROR zurück und plaziert einen Zeiger auf die Schnittstelle in *pif*.

Die Implementation von *QueryObject* in *TUnknown* liefert immer den Fehler HR_NOINTERFACE zurück. Klassen, die aus *TUnknown* abgeleitet sind, sollten diese Funktion überschreiben.

Beispiele für überschriebene Funktionen finden Sie im Quellcode für Klassen wie *TOcApp* und *TOcView*.

ThisUnknown

IUnknown& ThisUnknown();
Gibt eine Referenz auf die *IUnknown*-Schnittstelle für **this** zurück, nicht auf die äußeren oder inneren Objekte der Aggregation.

Geschütztes Datenelement

Outer

IUnknown* Outer;
Enthält einen Zeiger auf die Schnittstelle *IUnknown* des äußeren Objekts in einer Gruppe aggregierter Objekte.

Siehe auch: TUnknown::GetOuter, TUnknown::SetOuter

Klasse TXAuto ocf/autodefs.h

Basisklasse

TXBase
TXAuto ist das Exception-Objekt, das ObjectComponents absetzt, wenn es bei der Verarbeitung von Automatisierungsaufrufen auf einen unerwarteten Fehler trifft. Die möglichen Fehler werden von den geschachtelten Aufzählungswerten *TError* angezeigt.

Siehe auch: Klasse TXBase (OWL.HLP), Klasse TXObjComp, Klasse TXOle, Klasse TXRegistry

Öffentlicher Konstruktor

Konstruktor

TXAuto(TXAuto::TError err);
Konstruiert ein Exception-Objekt, das das in *err* angezeigte Problem beschreibt.

Siehe auch: Aufzählungstyp TXAuto::TError

Öffentliches Datenelement

ErrorCode

TError ErrorCode;
Enthält den Code, der das Problem identifiziert, für dessen Beschreibung das Objekt konstruiert wurde.

Siehe auch: Aufzählungstyp TXAuto::TError

Typdefinition

TError

Aufzählungstyp TError
Die Werte der Aufzählung identifizieren mögliche Fehler, die während der Automatisierung auftreten können.

Konstante	Bedeutung
xNoError	Kein Fehler aufgetreten.
xConversionError	Problem bei der Umwandlung eines Wertes aus einer VARIANT-Vereinigungsmenge in den erwarteten Datentyp.
xNotIDispatch	Versuch, einen Automatisierungsbefehl an ein Objekt zu schicken, das keine Befehle ausführt.
xForeignIDispatch	Versuch, einen Automatisierungsbefehl an ein automatisiertes Objekt zu schicken, das nicht von *TAutoProxy* abstammt.

xTypeMismatch	Ein zur Verfügung gestelltes Argument kann nicht in den erforderlichen Typ umgewandelt werden.
xNoArgSymbol	Ein Befehl hat versucht, mehr Argumente zu verwenden, als der Server erkennt.
xParameterMissing	Ein Automatisierungsaufruf konnte beim Setzen eines Eigenschaftswertes ein benötigtes Argument nicht zur Verfügung stellen.
xNoDefaultValue	Ein Parameter fehlt, und es wurde kein Standardwert zur Verfügung gestellt.
xValidateFailure	Der Code einer benutzerdefinierten Hook-Prozedur zur Gültigkeitsprüfung hat angezeigt, daß die von im empfangenen Argumentwerte nicht akzeptabel sind.

Klasse TXObjComp ocf/ocdefs.h

Basisklasse

TXBase
TXObjComp ist das Exception-Objekt, das ObjectComponents absetzt, wenn es bei der Abarbeitung seines eigenen internen Codes auf einen unerwarteten Fehler trifft. Die möglichen Fehler werden von den verschachtelten Aufzählungswerten *TError* angezeigt.

Siehe auch: Klasse TXAuto, Klasse TXBase, Klasse TXOle, Klasse TXRegistry

Öffentlicher Konstruktor

Konstruktor

TXObjComp(TXObjComp::TError err, const char* msg = 0);
Konstruiert ein Exception-Objekt, um das von *err* angezeigte Problem zu beschreiben. Verbindet den optionalen String *msg* mit dem Fehler.

Siehe auch: TXObjComp::TError

Öffentliche Elementfunktion

ErrorCode

TError ErrorCode
Enthält den Fehlercode, der das Problem identifiziert, zu dessen Beschreibung das Objekt konstruiert wurde.

Siehe auch: Aufzählungstyp TXObjComp::TError

Typdefinition

TError

enum TError
Die Werte der Aufzählung identifizieren mögliche Fehler, die innerhalb ObjectComponents auftreten können.

Konstante	Bedeutung
Anwendungsfehler	
xNoError	Es ist kein Fehler aufgetreten.
xBOleLoadFail	Die BOCOLE-Support-Bibliothek konnte nicht geladen werden.
xBOleBindFail	ObjectComponents konnte eine benötigte Schnittstelle nicht aus der BOCOLE-Support-Bibliothek erhalten.
xDocFactoryFail	TOcApp konnte die Anmeldung nicht bei OLE registrieren oder abmelden.
xRegWriteFail	Der Registrar konnte nicht in die Registrierdatenbank des Systems schreiben.
Dokument- und Teilfehler	
xMissingRootIStorage	Das Dokument, in dem ein Teil sich selbst konstruieren sollte, besitzt kein Stammspeicherabteilobjekt. (Ohne ein Speicherabteil besitzt das Dokument keinen Ort, an dem es seine Teile speichern kann.)
xInternalPartError	ObjectComponents konnte ein Teilobjekt nicht erzeugen.
xPartInitError	ObjectComponents konnte einen neu erzeugten Teil nicht initialisieren.
xDocSaveError	Ein *TOcDocument* konnte sich selbst nicht in eine Datei schreiben.
I/O-Fehler	
xStorageOpenError	Ein Dokument konnte sein Speicherateilobjekt nicht öffnen.
xStreamOpenError	Ein Dokument konnte das Stream-Objekt, das es zur Datei-I/O benötigt, nicht öffnen.
xStreamWQrite Error	Ein Dokument konnte nicht in das Stream-Objekt schreiben, das es zur Datei-I/O benötigt.

Klasse TXOle ocf/oleutil.h

Basisklasse

TXBase

TXOle ist das Exception-Objekt, das ObjectComponents absetzt, wenn es bei der Ausführung eines OLE API-Aufrufs auf einen unerwarteten Fehler trifft.

Die Methode *Check* des Objekts ist statisch, so daß Sie sie aufrufen können, ohne tatsächlich ein *TXOle*-Objekt zu erzeugen. Wenn die von Ihnen übergebenen Parameter das Auftreten eines Fehlers anzeigen, erzeugt Check ein *TXOle*-Objekt und setzt die Exception für Sie ab.

Siehe auch: Klasse TXAuto, Klasse TXObjComp, Klasse TXRegistry

Öffentliche Konstruktoren und Destruktor

Konstruktoren

Form 1

TXOle(const char far* msg, HRESULT stat);

Erzeugt ein OLE-Exception-Objekt. *msg* zeigt auf eine Fehlermeldung, und *stat* erhält den Rückgabewert von einem OLE API-Aufruf.

Form 2

TXOle(const TXOle& copy);
Konstruiert ein neues OLE-Exception-Objekt, indem es das als *copy* übergebene kopiert.

Normalerweise müssen Sie ein OLE-Exception-Objekt nicht direkt konstruieren. Rufen Sie statt dessen *Check* auf.

Destruktor

~TXOle();
Entfernt das *TXOle*-Objekt.

Siehe auch: TXOle::Check

Öffentliche Elementfunktionen

Check

Form 1

static void Check(HRESULT stat, const char far* msg);
Wenn *stat* einen Fehler anzeigt, setzt Form 1 eine *TXOle*-Exception ab, die den Fehlerstring *msg* enthält.

Form 2

static void Check(HRESULT stat);
Zeigt *stat* einen Fehler an, löst Form 2 eine *TXOle*-Exception aus, die den Fehlerstring enthält "OLE call FAILED, ErrorCode = *stat*". Dabei wird *stat* als achtstelliger Hexadezimalwert dargestellt.

Wenn Sie bei der Ausführung von Programmen auf diese Fehlermeldung stoßen, können Sie sie in der Datei OLE_ERRS.TXT nachschlagen. Zu Ihrer Bequemlichkeit sind die Fehlercodes den entsprechenden Kommentaren aus den OLE-Systemheader-Dateien zugeordnet.

Überprüft, ob ein Fehler aufgetreten ist und löst eine Exception aus, wenn das der Fall ist. *stat* ist der Rückgabewert eines OLE API-Aufrufs. *Check* ist statisch, so daß sie es aufrufen können, ohne vorher ein *TXOle*-Objekt zu erzeugen. Wenn *stat* einen Fehler anzeigt, erzeugt *Check* ein *TXOle*-Objekt und setzt eine Exception ab.

Öffentliches Datenelement

Stat

long Stat;
Stat ("status") enthält den Ergebniscode, der von einer OLE API zurückgegeben wird.

Klasse TXRegistry ocf/ocdefs. h

Basisklasse

TXBase
TXRegistry ist das Exception-Obvjekt, das ObjectComponents absetzt, wenn es beim Lesen oder Schreiben der Registrierdatenbank des Systems auf einen unerwarteten Fehler trifft.

Die Methode *Check* des Objekts ist statisch, so daß Sie sie aufrufen können, ohne tatsächlich ein *TXRegistry*-Objekt zu erzeugen. Wenn die von Ihnen übergebenen Parameter das Auftreten eines Fehlers anzeigen, erzeugt Check ein *TXRegistry*-Objekt und setzt die Exception für Sie ab.

Siehe auch: Klasse TXAuto, Klasse TXObjComp, Klasse TXOle

Öffentliche Konstruktoren

Konstruktoren

Form 1

TXRegistry(const char* msg, const char* key);
Erzeugt ein Registrier-Exception-Objekt. *msg* zeigt auf eine Fehlermeldung, und *key* zeigt auf den Namen des Schlüssels in der Regstrierdatenbank, den ObjectComponents bearbeitete, als die Exception auftrat.

Form 2

TXRegistry(const TXRegistry& copy);
Der Kopierkonstruktor erzeugt ein neues Registrier-Exception-Objekt, indem er das in *copy* übergebene kopiert.

Normalerweise brauchen Sie Registrier-Exceptions nicht direkt zu konstruieren. Rufen Sie stattdessen *Check* auf.

Siehe auch: TXRegistry::Check

Öffentliche Elementfunktionen

Check

static void Check(long stat, const char* key);
Überprüft den Wert von *stat* um festzustellen, ob ein Fehler aufgetreten ist, und löst in diesem Fall eine Exception aus. *stat* ist der Rückgabewert eines Registrierbefehls. *key* ist der Name des Schlüssels, den der Registrierbefehl gerade bearbeitete.

Check ist statisch, so daß Sie es aufrufen können, ohne zuvor ein *TXRegistry*-Objekt zu erzeugen. Wenn *stat* ungleich 0 ist, dann erzeugt *Check* ein *TXRegistry*-Objekt und löst eine Exception aus. Die Exception enthält den Meldungs-String "Registry failure on Key:*key*, ErrorCode = *stat*".

Key

const char* Key;
Zeigt auf den Namen des Registrierschlüssels, den ObjectComponents gerade bearbeitete, als die Exception eintrat.

Registrierschlüssel typehelp

Registriert den Namen einer Hilfedatei (.HLP), die Informationen über die Methoden und Eigenschaften enthält, die Ihr Programm für die Automatisierung zur Verfügung stellt. Wenn die Datei sich nicht im gleichen Verzeichnis befindet wie das Programm, müssen Sie auch darauf achten, *helpdir* zu registrieren.

typehelp ist in der Anwendungsregistriertabelle eines Automatisierungs-Servers gültig. Er ist optional. Ferner kann der Dateiname lokalisiert werden, so daß auf einfache Weise verschiedene Hilfedateien für verschiedene Sprachen zur Verfügung gestellt werden können.

Zum Registrieren von *typehelp* wird das Makro REGDATA verwendet. *typehelp* wird als erster Parameter übergeben und der Dateiname als zweiter.

Siehe auch: Registrierschlüssel helpdir, Makro REGDATA, Registrierschlüssel typelib

Registrierschlüssel usage

Legt fest, ob Ihre Anwendung mit einer einzigen Instanz mehrere Benutzer unterstützen darf, oder ob für jeden neuen OLE-Client ein neues Exemplar gestartet werden soll. Der Kommandozeilenschalter *-Automation* überschreibt diese Einstellung und erzwingt den Einzelbetrieb, wenn ein Automatisierungs-Server aufgerufen wird.

Der Schlüssel *usage* ist in allen Server-Registriertabellen gültig. Er ist immer optional. Wenn Sie ihn weglassen, registriert ObjectComponents die Anwendung standardmäßig für die Unterstützung von reinem Einzelbetrieb.

Zum Registrieren des *usage*-Schlüssels wird das Makro REGDATA verwendet. *usage* wird als erster Parameter übergeben und eine der *ocrxxxx*-Usage-Konstanten als zweiter.

REGDATA(usage, ocrSingleUse) // ein Client pro Instanz (Standard)

Siehe auch: ocrxxxx-Usage-Konstanten, Makro REGDATA

Registrierschlüssel verb*n*

Ein String, der eine Aktionsart benennt, die der Server mit seinen Objekten ausführen kann. Container fügen die Aktionsarten des aktiven Objekts ihrem Menü Bearbeiten hinzu.

verb0 ist der Name der Hauptaktionsart (Voreinstellung) für die Klasse. Die Hauptaktionsart wird ausgeführt, wenn der Anwender einen Doppelklick auf das Objekt ausführt. Verwenden Sie *verb1* bis *verb7*, um zusätzliche Aktionsarten zu registrieren. Die Konstante *ocrVerbLimit*, die in ocf/ocreg.h definiert ist, repräsentiert die Höchstzahl zulässiger Aktionsarten (8).

Die Schlüssel *verbn* sind in den Dokumentregistriertabellen eines Servers, der Verknüpfung und Einbettung unterstützt, gültig. Jeder Server sollte eine Standardaktionsart registrieren. Andere Aktionsarten sind optional.

Verwenden Sie zum Registrieren einer Aktionsart das Makro REGDATA und übergeben Sie *verbn* als ersten Parameter und einen String für den Menüeintrag als zweiten Parameter.

```
REGDATA(verb0, "&Edit")   // Standardaktionsart
REGDATA(verb1, "&Open")   // eine andere mögliche Aktionsart (optional)
```

Siehe auch: Makro REGDATA, Registrierschlüssel verb*n*opt

Registrierschlüssel verb*n*opt

Registriert Options-Flags, die die Aktionsarten des Servers beschreiben. Die Flags legen fest, wie die Aktionsarten im Menü des Containers erscheinen. Sie können beispielsweise grau dargestellt oder deaktiviert sein.

Aktionsart-Optionen sind in den Dokument-Registriertabellen jedes Verknüpfungs- und Einbettungs-Servers gültig. Sie sind immer optional. Aktionsart-Optionen haben keine Bedeutung, wenn nicht auch Aktionsarten registriert werden.

Zum Registrieren von Aktionsarten verwenden Sie das Makro REGVERBOPT. Übergeben Sie den Schlüssel für eine Aktionsart (wie etwa *verb0* oder *verb1*) als ersten Parameter. Verwenden Sie als zweiten Parameter *ocrxxxx*-Aktionsart-Menükonstanten, als dritten *ocrxxxx*-Aktionsart-Attributkonstanten.

```
REGVERBOPT(verb2, ocrGrayed, ocrOnContainerMenu | ocrNeverDirties)
```

Siehe auch: ocrxxxx-Aktionsart-Menükonstanten, ocrxxxx-Aktionsart-Attributkonstanten, Makro REGVERBOPT, Registrierschlüssel verb*n*.

Registrierschlüssel version

Registriert einen Versions-String für die Anwendung und die Typbibliothek. Der String kann Unterversionsnummern enthalten, die durch Punkt abgetrennt sind. OLE ignoriert Versionsnummern nach den ersten zwei (Haupt- und Unterversionsnummer).

Der Versionsschlüssel ist in jeder Registriertabelle gültig. Er ist immer optional.

Um *version* zu registrieren, verwenden Sie das Makro REGDATA und übergeben Sie *version* als ersten Parameter und den Versionsnummer-String als zweiten.

REGDATA(version, "1.0.5")

Siehe auch: Registrierschlüssel description, Registrierschlüssel permid, Registrierschlüssel permname

Botschaft WM_OCEVENT ocf/ocapp.h

ObjectComponents definiert die Botschaft WM_OCEVENT, um das Fenster einer Anwendung zu benachrichtigen, wenn wichtige Ereignisse eintreten, die mit OLE zusammenhängen.

Botschaft	Bedeutung
WM_OCEVENT	Benachrichtigung über ein OLE-Ereignis von ObjectComponents. Der *wParam*-Wert identifiziert das betreffende Ereignis.

Siehe auch: Botschaften OC_APPxxxx, Botschaften OC_VIEW

K A P I T E L 7

Übersicht über ObjectSupport

Dieses Kapitel gibt einen Überblick über die Klassen von Object Support, Bibliotheken und Header-Dateien. Diese Klassen unterstützen verschiedene Dienste, die Ihnen helfen, Ihre ObjectWindows-Anwendung zu gestalten. Diese Klassen schließen folgende Gruppen mit ein:

- Mathematische Klassen wie *TPoint*, *TSize* und *TRect* , welche die Bildschirmkoordinaten und die Abmessungen von Rechtecken definieren.
- Klassen für Dokumentschablonen, welche die Gestaltung der Dok/View-Anwendungen erleichtern.
- *TLocaleString*,welche die Registrierinformationen von OLE lokalisiert, die für Container und Server erforderlich sind.

Folgende Tabelle listet die Dateien auf, welche die Bibliothek Object Support enthält (Verzeichnis ..\OSL).

Tabelle 7.1: Auflistung der Bibliotheksdateien von Object Support

Dateiname	Beschreibung
geometry.h	Enthält Beschreibungen von mathematischen Klassen wie *TPoint*, *TRect*, *TSize*, *TResId*, *TDropInfo*, *TProcInstance* und *TPointer*.
doctpl.h	Enthält Definitionen der Dateien *TDocTemplate* und *TDocTemplateT*<D,V>.
defs.h	Enthält allgemeine Definitionen, einschließlich der Definitionen in windows.h und behandelt den Datentyp BOOL.
locale.h	Definiert die Klasse *TLocaleString* ebenso wie Registriermakros.

K A P I T E L 8

ObjectSupport-Referenz

TLangId typedef — locale.h

```
typedef unsigned short TLangId;
```

Enthält eine Sprachen-ID, d.h. eine vordefinierte Zahl, die eine Basissprache und einen Dialekt repräsentiert. Die Zahl 409 zum Beispiel repräsentiert amerikanisches Englisch. *TLocaleString* verwendet die Sprachen-ID, um die korrekte Übersetzung für Strings zu ermitteln.

Siehe auch TLocaleString

Registriermakros

Folgende, in *locale.h* definierte Makros nehmen Rücksicht auf die verschiedenen OLE Registrierprozeduren. Diese Makros vereinfachen den Bildungsprozess einer Registriertabelle (einer speziellen Art Nachschlagetabelle) für einen automatischen Server und für eine nicht automatisierte Anwendung oder ein Dokument, die mit einem Container oder Server verbunden sind. Eine Sammlung von wichtigen Statistiken über ein Objekt, die Registriertabelle, versieht eine externe Beschreibung mit dem assoziierten Objekt. Einige Informationen fließen in das Systemregister ein und werden von OLE verwendet. Einige Informationen werden in dem Dialogfenster *Datei/Öffnen* angezeigt, wenn der Benutzer die Funktion *Objekt einfügen* auswählt.

Die Registriermakros bauen eine Struktur *TRegList* auf, welche Einträge vom Typ *TRegItem* enthält und wie folgt definiert ist:

```
struct TRegItem {
  char*  Key;              //Name des Eintrags
  TLocaleString Value;    //String-Wert für den Eintrag
};
```

Wenn Sie diese Makros verwenden, brauchen Sie die Strukturen *TRegItem* und *TRegList* nicht selbst zu bilden.

Obwohl Server und Container dieselben Makros verwenden, geben sie verschiedene Arten von Information an die Registrierungsstrukturen weiter. Abhängig von der Menge der Information, welche Sie Ihrer Anwendung zuordnen, und ob Sie eine Struktur mit Dokument- oder Anwendungsinformation erstellen möchten, verwenden Sie eine oder mehrere dieser Registriermakros.

Registriermakros

Makro	Bedeutung
BEGIN_REGISTRATION	Startet eine Registriermakrotabelle
END_REGISTRATION	Beendet eine Registriermakrotabelle
REGDATA	Registriert Information über die Anwendung, z.B. Klassen-ID, Beschreibung, Dokumentfilter und Debugger
REGDOCFLAG	Registriert eine Reihe von Dokumentenflags. Erforderlich für eine Dokumentregistriertabelle
REGFORMAT	Registriert ein Datenformat
REGICON	Registriert ein Symbol
REGITEM	Registriert ein angepaßtes Format
REGSTATUS	Registriert einen Aspektstatus
REGVERBOPT	Registriert eine Option für eine Aktionsart
REGISTRATION_FORMAT_BUFFER	Setzt die Größe des Pufferspeichers für die benötigte Erweiterung

REGDATA, das wichtigste der in der Registriertabelle benutzten Makros, leitet die String-Daten in ihre Argumente. Die Makros REGFORMAT, REGDOCFLAG, REGICON, REGSTATUS und REGVERBOPT formatieren numerische Werte als String-Werte.

Suchen Sie im *ObjectWindows Programmierhandbuch* nach der Liste mit Elementnamen (referenziert als Schlüssel), die in der Anwendung- und Dokumentregistriertabelle benötigt werden. Information darüber, wie man diese Makros in den OLE-fähigen Anwendungen benutzt, sehen Sie in den Abschnitten »Registrierung eines verknüpften und eingebetteten Servers« und »Registrierung des Containers« im *ObjectWindows Programmierhandbuch*. Die Beispielanwendungen REGTEST.CPP und STEP15DV.CPP auf der beigefügten Diskette versorgen Sie mit Beispielen für Registriertabellen, ausgelegt für die unterschiedlichsten Arten von Anwendungen und Dokumenten.

Siehe auch END_REGISTRATION-Makro, BEGIN_REGISTRATION-Makro, *TLocaleString*

BEGIN_REGISTRATION-Makro locale.h

```
BEGIN_REGISTRATION(regname)
```

Zeigt den Beginn einer Registriermakrotabelle an. Das Makro nimmt ein Argument (*regname*), welches der Name der zu bildenden Struktur ist. Innerhalb des Registriertabellenmakros gibt es mehrere Makros, die die Registrierungsstruktur bilden. Abhängig vom Typ der Anwendung oder des Dokuments werden unterschiedliche Makros verwendet. Das folgende Beispiel aus STEP15.CPP registriert den Zeichenstift als Server-Anwendung und bildet eine Struktur *AppReg*:

```
BEGIN_REGISTRATION(AppReg)
  REGDATA(clsid,       "{5E4BD320-8ABC-101B-A23B-CE4E85D07ED2}")
  REGDATA(description,"OWL Drawing Pad Server")
END_REGISTRATION
```

Siehe auch END_REGISTRATION-Makro, REGDATA-Makro,

END_REGISTRATION-Makro locale.h

```
END_REGISTRATION
```

END_REGISTRATION Zeigt das Ende einer Registriermakrotabelle an. Man kann die Registriermakros in die BEGIN_REGISTRATION- und END_REGISTRATION-Makros einfügen,um eine Registrierungsstruktur zu bilden.

Siehe auch BEGIN_REGISTRATION-Makro

REGDATA-Macro locale.h

```
REGDATA(var,val)
```

Das Hauptregistriermakro REGDATA registriert Informationen einer Anwendung oder eines Dokuments.Das Makro benötigt immer eine Option (zum Beispiel:*clsid*) und einen dazugehörigen Wert (zum Beispiel:5E4BD320-8ABC-101B-A23B-CE4E85D07ED2).Das folgende Beispiel aus STEP15.CPP Ihrer mitgelieferten Diskette übergibt die Klassen-ID und -Beschreibung, um eine Registrierungsstruktur der Anwendung *AppReg* zu bilden:

```
BEGIN_REGISTRATION(AppReg)
  REGDATA(clsid,       "{5E4BD320-8ABC-101B-A23B-CE4E85D07ED2}")
  REGDATA(description,"OWL Drawing Pad Server")
END_REGISTRATION
```

Die Registrierungsstruktur für einen automatisierten Server enthält folgende REGDATA-Makros:

```
BEGIN_REGISTRATION(myappreg)
 REGDATA(clsid,       "{01234567-1234-5678-1122-334455667788}")
 REGDATA(progid,      "MySample")
 REGDATA(description,"My Sample 1.0 Application")
 REGDATA(cmdline,     "/automation")
 REGDATA(version,     "1.2")
END_REGISTRATION
```

Jedes automatisierbare Objekt erfordert eine Programm-ID und ein Kommandozeilenargument, das in die Kommandozeile des Servers eingefügt wird. Gewöhnlich wird nur eine Klassen-ID für die Anwendung definiert. Im *ObjectWindows Programmierhandbuch* finden Sie weitere Informationen darüber, wie ein automatisierter Server registriert wird.

Für jede Registrierungsstruktur einer Anwendung dürfen eine oder mehrere Registrierungsstrukturen eines Dokuments erstellt werden. Das folgende Beispiel benutzt Daten von verschiedenen REGDATA-Makros, um eine Registrierungsstruktur *DocReg* zu bilden.

```
BEGIN_REGISTRATION(mydocreg)
 REGDATA(description,"My Sample 1.0 Document")
 REGDATA(extension,
"myd") //Verwenden Sie keinen Punkt vor der Erweiterung.
 REGDATA(directory,  "C:\temp")
 REGDATA(docfilter,  "*.drw;*.drx")
 ...//Hier°zusätzliche°Makros°einfügen.
 REGDATA(debugger,   "tdw -t") // Setzen der°Debugger- Optionen
 REGDATA(progid,     "MyDocument") //Nur für Server
 REGDATA(menuname,   "My Document") //Nur für Server
 REGDATA(insertable,0) // Nur für Server
 REGDATA(usage,    ocrMultipleUse) // Nur für Server
 REGDATA(verb0,      "&Edit") //Nur für Server
 REGDATA(verb1,      "&Open") //Nur für Server
 REGDATA(verb2,      "&Play") //Nur für Server
 ...//Hier°zusätzliche°Makros°einfügen.
END_REGISTRATION
```

Siehe auch BEGIN_REGISTRATION-Makro, END_REGISTRATION-Makro

REGITEM-Makro locale.h

```
REGITEM(key,val)
```

Das REGITEM-Makro dient zum Schreiben angepasster Einträge für die Systemregistratur. Das folgende Beispiel aus REGTEST.CPP registriert Konvertierungsformate.

```
REGITEM("CLSID\\>clsid<\\Conversion\\Readable\\Main",°°"FormatX,FormatY")
```

Der erste String ist der Registrierschlüssel mit einem Parameter (<*clsid*>). Wenn die Registrierinformation erstellt worden ist, ersetzt ein aktueller Wert die Template-Parameter.

Siehe auch BEGIN_REGISTRATION-Makro, END_REGISTRATION-Makro

REGFORMAT-Makro locale.h

```
REGFORMAT(i,f,a,t,d)
```

Das REGFORMAT-Makro zeigt Datenformate an, die der Server oder Container unterstützt, und hat die folgenden Parameter:

Parameter	
i	Prioritätsordnung für das ausgewählte Datenformat, mit 0 für höchsten Vorrang
f	Datenformat, z.B. *ocrText*, *ocrTiff*, *ocrDib* usw.
a	Format zum Darstellen der Daten
t	Medium für die Datenübertragung, z.B. , *ocrMfPict* (METAFILEPICT Struktur), *ocrGDI* (GDI Object, wie eine Bitmap), *ocrIStream* (Stream-Objekt in einer zusammengesetzter Datei) usw.
d	Ob die Daten im gekennzeichneten Format gesendet bzw. empfangen wurden oder nicht . Akzeptierte Werte sind *ocrGet* (importiert Daten im spezifizierten Format), *ocrSet* (exportiert Daten im spezifizierten Format), oder *ocrGetSet* (exportiert und importiert Daten im spezifizierten Format).

STEP15DV.CPP, z.B. registriert das folgende Zwischenablageformat:

```
REGFORMAT(1, ocrMetafilePict, ocrcontent, ocrMfPict, ocfSet)
```

und erzeugt den String "format1, 3, 1, 1056,1." Obwohl Sie den String der Zahlen eingeben könnten, ist es viel leichter, aufzählende Werte zu verwenden. Siehe *ocrxxxx Zwischenablage*-Konstanten zur Beschreibung der akzeptierten Datenformate.

Um eine Registrierungsstruktur zu bilden, benutzt man REGFORMAT innerhalb von BEGIN_REGISTRATION- und END_REGISTRATION-Makros in einer Registriermakrotabelle. Alle registrierten Formate, die REGFORMAT benutzt haben, werden auch automatisch in der Windows Zwischenablage registriert. Sie können Ihre eigenen Formate registrieren, indem Sie einen String einfügen, der Ihr eigenes Format anzeigt. Zum Beispiel:

```
REGFORMAT(2, "ANewFormat", ocrContent, ocrIStorage, ocrGetSet)
```

Um Namen für Ihre eigenen Formate vorzusehen, rufen sie *TOleFrame::AddUserFormatName* auf. Diese Funktion ordnet ein Zwischenablagedatenformat einer Beschreibung des Datenformats zu, wie es den Benutzern in dem Hilfetext des Dialogfensters *Paste Special* erscheint.

Siehe auch BEGIN_REGISTRATION-Makro, *ocrxxxx* Zwischenablage-Konstanten, *ocrxxxx* Medium-Konstanten, *TOleFrame::AddUserFormatName*

REGSTATUS-Makro — locale.h

```
REGSTATUS(a,f)
```

Das REGSTATUS-Makro zeigt, in welcher Weise sich die Sicht (referenziert als »aspect«) eines Objektes verhält, und besitzt folgende Argumente:

Argument	
a	Inhalt des Objektes
f	Einer der in *Objektstatus ocrxxxx* aufgezählten Werte, zeigt den Status des Objektes an, z.B. *ocrOnlyIconic*, *ocrActivateWhenVisible* usw.

Das Objekt kann als Träger vieler unterschiedlicher Aspekte des Verhaltens definiert werden. Ein Objekt, das als *ocrActivateWhenVisible* registriert ist, ist aktiv, wann immer es sichtbar ist.

Server, welche gelinkte und eingebettete Objekte unterstützen, benutzen dieses Makro, um das Verhalten zu registrieren, welches ein Objekt ausstellt, wenn es sichtbar gemacht wird. Dieses Verhalten ist referenziert als Aspektstatus oder einfach Objektaspekt.

Das Beispielprogramm REGTEST.CPP auf der beigefügten Diskette enthält die folgenden REGSTATUS-Makros:

```
REGSTATUS(all, ocrNoSpecialRendering)
REGSTATUS(icon, ocrOnlyIconic)
```

Das erste Makro registriert Flags für alle Aspekte des Objekts, während das zweite Makro Flags für den Symbolaspekt des Objektes registriert. (Das im REGSTATUS-Makro benutzte Symbol muß definiert und registriert sein.)

Siehe auch BEGIN_REGISTRATION-Makro, REGICON-Makro, *ocrxxxx* Objektstatus Konstante

REGVERBOPT-Makro — locale.h

```
REGVERBOPT(v,mf,sf)
```

Registriert die Aktionen, die ein Server auf seine Objekte ausführen kann. Die Argumente kontrollieren das Erscheinen der Aktionsarten auf dem Container-Menü. Das Makro hat die folgenden Argumente:

Argument	
v	der Aktionsartschlüssel, z.B. *verb1* oder *verb2*.
mf	Ein Wert, der beschreibt, wie eine Server-Aktionsart auf dem Container-Menü erscheint. Dieser Wert muß eine der *ocrxxxx Aktionsart-Menü*-Konstanten sein, z.B., *ocrGrayed*, die die Aktionsart grau auf dem Bildschirm erscheinen lässt und die Aktionsart sperrt.
sf	Ein Wert, mit dem der Container Informationen erhält, wie er die Aktionsart benutzen soll. Dieser Wert muß einer der *ocrxxxx Aktionsart-Attribut*-Konstanten sein, z.B., *ocrNeverDirties*, die anzeigen, daß die Aktionsart niemals das Objekt modifiziert. Diese Optionen können ORed zusammen sein.

Das Beispielprogramm REGTEST.CPP auf der beigefügten Diskette schließt folgendes REGVERBOPT-Makro ein:

```
REGVERBOPT(verb1, ocrGrayed, ocrOnContainerMenu | ocrNeverDirties)
```

Diese Aktionsartoptionen sind optional und nur gültig, falls die Aktionsart in der Dokumentregistertabelle der Serverapplikation registriert ist. Um die Aktionsart zu registrieren, benutzen Sie

```
REGDATA(verb1, "&Open")
```

Siehe auch BEGIN_REGISTRATION Macro, *ocrxxxx* Aktionsart-Attribut-Konstanten, *ocrxxxx* Aktionsart-Menü-Konstanten

REGICON-Makro locale.h

```
REGICON(i)
```

Registriert ein Symbol, so daß das Objekt als Symbol angezeigt wird. Das Beispielprogramm REGTEST.CPP auf der beigefügten Diskette enthält das folgende REGICON-Makro:

```
REGICON(1)
```

Das Makro benötigt ein Argument; der Index des Defaultsymbols ist zu benutzen. Dieses Argument zeigt an, welches Symbol von der Ressourcendatei ermittelt werden muß, wenn das Dokument als Symbol dargestellt werden soll.

Siehe auch BEGIN_REGISTRATION-Makro

REGDOCFLAGS-Makro locale.h

```
REGDOCFLAGS(i)
```

Zeigt Optionen für das Dokument und definiert die Eigenschaften der Dokument-Templates. Die REGDOCFLAGS-Argumente teilen dem Dokumentverwalter mit ,wie er die Dokumente und Ausschnitte anzeigen und behandeln soll. Obwohl Sie diese Information aus Gründen der Rückwärtskompatibilität noch zu der Dokumentschablone leiten können, die separate Parameter in dem Konstruktor benutzt, sollten neuere Programme die REGFORMAT, REGDOCFLAGS, und REGDATA-Makros benutzen, um ein Objekt des Dokument-Templates zu erzeugen.

Das Beispielprogramm REGTEST.CPP auf der beigefügten Diskette enthält die folgende REGDOCFLAGS-Makrodeklaration innerhalb der Struktur der Dokumentregistrierung:

```
BEGIN_REGISTRATION(mytplreg)
 REGDATA(description,"My Sample Draw View")
 REGDATA(filter,     "*.drw;*.drx")
 REGDATA(defaultext, "dvw")
 REGDATA(directory,  0)
 REGDOCFLAGS(dtAutoDelete | dtUpdateDir | dtCreatePrompt)
END_REGISTRATION
```

Die Argumente zu REGDOCFLAGS definieren die Eigenschaften des Dokuments. Im Fall von *dtAutoDelete* ist das Dokument beispielsweise gelöscht, wenn der letzte Ausschnitt gelöscht ist.:

Bestimmte Dokumente müssen mit verschiedenen Optionen registriert werden. Um eine Doc/View-Anwendung mit einem Objekt *TOleDocument* zu registrieren, müssen Sie die *dtRegisterExt*- und *dtAutoOpen*-Flags spezifizieren. Wenn es sich um eine Doc/View-Anwendung handelt und die Dokumentschablone nicht versteckt ist, müssen Sie Beschreibung, Filter und Erweiterung registrieren.

Siehe auch BEGIN_REGISTRATION-Makro, REGFORMAT-Makro, REGDATA-Makro, dt Dokument-Template-Konstante

REGISTRATION_FORMAT_BUFFER-Makro locale.h

```
REGISTRATION_FORMAT_BUFFER(n);
```

Reserviert Speicher im Hauptspeicher (*n*) für die Erweiterung der Werte, die an die Registrierungsmakros übergeben worden sind, welche das Formatieren eines Dokuments oder einer Anwendung kontrollieren. Gewöhnlich erlaubt man 10 Bytes für jeden in das REGFORMAT-Makro

geleiteten Wert zusätzlich zum Speicherbedarf, der erforderlich ist für Strings, die in die REGSTATUS-, REGVERBOPT-, REGICON-, oder REGFORMAT-Makros geleitet werden. Das Beispielprogramm STEP15.CPP benutzt beispielsweise dieses Makro, um 100 Bytes Speicherbedarf zu deklarieren:

```
REGISTRATION_FORMAT_BUFFER(100);
```

Siehe auch REGFORMAT-Makro, REGSTATUS-Makro

Klasse TDocTemplate — doctpl.h

TDocTemplate ist eine abstrakte Basisklasse, die die Funktionalität für ein Dokument-Template enthält. *TDocTemplate*-Klassen erzeugen Dokumente und Views von Ressourcen und behandeln das Benennen und Durchsehen eines Dokumentes. Die Dokumentverwaltung verwaltet eine Liste der aktuellen Template-Objekte. Jeder Dokumenttyp erfordert ein separates Dokument-Template.

Öffentliche Elementfunktionen

ClearFlag

```
void ClearFlag(long flag);
```

Löscht eine Dokument-View-Konstante.

Siehe auch: *dtxxxx* Dokument-View-Konstanten

Clone

```
virtual TDocTemplate* Clone(TModule* module, TDocTemplate*&
phead=DocTemplateStaticHead)=0;
```

Macht eine Kopie eines Dokument-Templates.

ConstructDoc

```
virtual TDocument* ConstructDoc(TDocument* parent = 0) = 0;
```

Eine rein virtuelle Funktion, die in einer abgeleiteten Klasse definiert werden muß. *ConstructDoc* erzeugt ein Element, das durch die Klasse des Dokument-Templates spezifiziert wird. Diese Funktion wird anstelle von *CreateDoc* verwendet.

Siehe auch: *TDocManager::CreateDoc*

ConstructView

```
virtual TView* ConstructView(TDocument& doc = 0) = 0;
```

Eine rein virtuelle Funktion, die in einer abgeleiteten Klasse definiert werden muß. *ConstructView* erzeugt die Darstellung, die in der Klasse des Dokument-Templates spezifiziert wird.

Siehe auch: *TDocManager::CreateView*

CreateDoc

```
virtual TDocument* CreateDoc(const char far* path, long flags = 0)= 0;
```

Eine abstrakte Funktion, die in einer abgeleiteten Klasse definiert werden muß. *CreateDoc* erzeugt ein Dokument, basierend auf dem Verzeichnis-

pfad (*path*), dem angegebenen Template und dem Wert *flags*. Wenn *path* Null ist und das neue Flag (*dtNewDoc*) nicht gesetzt ist, wird das Dialogfenster angezeigt. Diese Funktion ist veraltet, verwenden Sie statt dessen die Funktion *ConstructDoc*.

Siehe auch: *TDocManager::CreateAnyDoc*, *TDocTemplate::ConstructDoc*

CreateView

```
virtual TView* CreateView(TDocument& doc, long flags) = 0;
```

Eine abstrakte Funktion, die in einer abgeleiteten Klasse definiert werden muß. *CreateView* erzeugt das View, das durch die Dokument-Template-Klasse angegeben ist. Diese Funktion ist veraltet, verwenden Sie statt dessen die Funktion *ConstructView*.

Siehe auch: *TDocManager::CreateAnyView*, *TDocTemplate::ConstructView*

GetDefaultExt

```
const char far* GetDefaultExt() const;
```

Ermittelt die Standard-Dateinamenserweiterung, die benutzt wird, wenn der Benutzer den Dateinamen ohne jede Erweiterung eingegeben hat. Wenn es keine Standarderweiterung gibt, enthält *GetDefaultExt* Null.

GetDescription

```
const char far* GetDescription() const;
```

Liefert die Template-Beschreibung, um sie im Listenfeld zur Dateiauswahl oder im Listenfeld zur *Datei/Neu*-Menüauswahl abzulegen.

GetDirectory

```
const char far* GetDirectory() const;
```

Liefert den Verzeichnispfad, der bei der Suche nach übereinstimmenden Dateien benutzt wird. Dieser wird aktualisiert, wenn eine Datei ausgewählt wird und das Flag *dtUpdateDir* gesetzt ist.

Siehe auch: *dt* Dokument-View-Konstanten

GetDocManager

```
TDocManager* GetDocManager() const;
```

Zeigt auf die Dokumentverwaltung.

GetFileFilter

```
const char far* GetFileFilter() const;
```

Liefert alle gültigen Dokument-Übereinstimmungsmasken, die bei der Suche nach Dateien benutzt werden.

GetFlags

```
long GetFlags() const;
```

Liefert die Dokument-Template-Konstanten, die anzeigen, wie das Dokument erzeugt und geöffnet wird.

Siehe auch: *dtxxxx* Dokument-Template-Konstanten

GetModule

```
TModule*& GetModule();
```

Gibt einen Zeiger auf ein Modul zurück.

GetNextTemplate

```
TDocTemplate* GetNextTemplate() const;
```

Gibt das nächste Template aus der Liste von Templates zurück.

GetRegList

```
TRegList& GetRegList() const;
```

Gibt die Registriertabelle des Programms zurück. Diese enthält die gegenwärtige Programm-ID, die Klassen-ID, den ausführbaren Pfad sowie andere Attribute, die verwendet werden, um ein Objekt *TDocTemplate* zu erzeugen. Weitere Informationen finden Sie in dem Eintrag für Registriermakros in diesem Handbuch, der darüber Auskunft gibt, wie die Registriermakros die Registrierinformationen generieren.

GetViewName

```
virtual const char far* GetViewName() = 0;
```

Eine abstrakte Funktion, die in einer abgeleiteten Klasse definiert werden muß. *GetViewName* liefert den Namen des mit dem Template assoziierten View.

InitDoc

```
TDocument* InitDoc(TDocument* doc, const char far* path, long flags);
```

InitDoc wird nur von der Unterklasse aufgerufen, so daß *CreateDoc* mit der Ausführung seiner Dokumente fortfahren kann.

Siehe auch: *TDocTemplate::CreateDoc*

InitView

```
TView* InitView(TView* view);
```

Wird ausschließlich von der Unterklasse verwendet, um mit der Ausführung von *CreateView* fortzufahren.

Siehe auch: *TDocTemplate::CreateView*

IsFlagSet

```
BOOL IsFlagSet(long flag);
```

Gibt einen Wert ungleich Null zurück, wenn das Dokument-Template-Flag gesetzt ist.

Siehe auch: *dtxxxx* Dokument-Template-Konstanten

IsMyKindOfDoc

```
virtual TDocument* IsMyKindOfDoc(TDocument& doc)=0;
```

Eine abstrakte Funktion, die in einer abgeleiteten Klasse definiert werden muß. *IsMyKindOfDoc* testet, ob das Template zur selben Klasse wie das Dokument oder zu einer abgeleiteten Klasse gehört.

Siehe auch: *TDocTemplateT::IsMyKindOfDoc*

IsMyKindOfView

```
virtual TView* IsMyKindOfView(TView& view) = 0;
```

Eine rein virtuelle Funktion, die in einer abgeleiteten Klasse definiert werden muß. *IsMyKindofView* testet, ob das Template zu der gleichen Klasse wie View oder zu einer abgeleiteten Klasse gehört.

IsStatic

```
bool IsStatic();
```

Gibt **true** zurück, wenn das Template statisch konstruiert wurde.

IsVisible

```
BOOL IsVisible();
```

Zeigt an, ob das Dokument im Dialogfenster *Dateiauswahl* angezeigt werden kann. Ein Dokument ist sichtbar, wenn *dtHidden* nicht gesetzt ist und wenn *Description* nicht Null ist.

SelectSave

```
BOOL SelectSave(TDocument& doc);
```

Fordert den Benutzer auf, einen Dateinamen für das Dokument auszusuchen. Filtert Dateien für reinen Lesezugriff heraus.

SetDefaultExt

```
void SetDefaultExt(const char far*);
```

Setzt die Standard-Dateinamenserweiterung, die benutzt wird, wenn der Benutzer den Dateinamen ohne jede Erweiterung eingegeben hat. Wenn es keine Standarderweiterung gibt, enthält *SetDefaultExt* Null.

SetDirectory

```
void SetDirectory(const char far*, int len);
```

Setzt den Verzeichnispfad, der bei der Suche nach übereinstimmenden Dateien benutzt wird. Dieser wird aktualisiert, wenn die Datei ausgewählt ist und das Flag *dtUpdateDir* gesetzt ist.

Siehe auch: *TDocTemplate::GetDirectory*

SetDocManager

```
void SetDocManager(TDocManager* dm);
```

Setzt die aktuelle Dokumentverwaltung auf das Argument *dm*.

SetFileFilter

```
void SetFileFilter(const char far*);
```

Setzt die gültige Dokument-Übereinstimmungsmaske, die bei der Suche nach Dateien benutzt werden soll.

SetFlag

```
void SetFlag(long flag);
```

Setzt die Dokument-Template-Konstanten, die anzeigen, wie das Dokument erzeugt und geöffnet wird.

Siehe auch: *dtxxxx* Dokument-Template-Konstanten

SetModule

```
TModule*& SetModule();
```

Setzt einen Modul-Zeiger.

Geschützte Konstruktoren und Destruktoren

Konstruktor

```
TDocTemplate(TRegList& regList, TModule*& module, TDocTemplate*& phead):
```

Verwendet die Informationen in der Registriertabelle, um ein *TDocTemplate*-Objekt mit der angegebenen Dateibeschreibung (*desc*), der Dateifiltermaske (*filt*), dem Suchpfad für die Ansicht des Verzeichnisses (*dir*), der Standard-Dateinamenserweiterung (*ext*) und mit Flags (*flags)* zu konstruieren, die die in der Registrierungsliste enthaltenen Optionen für die Erzeugung und die Ansicht repräsentieren. Setzt *Description* auf *desc*,

FileFilter auf *filt*, *Directory* auf *dir*, *DefaultExt* auf *ext* und *Flags* auf *flags*. Anschließend fügt sie dieses Template der Template-Liste der Dokumentverwaltung zu. Wenn die Dokumentverwaltung noch nicht konstruiert ist, fügt sie das Template einer statischen Liste hinzu, die die Dokumentverwaltung ihrer Template-Liste später hinzufügen wird.

Das Argument *module* spezifiziert das *TModule* des Aktivators. *phead* spezifiziert den Templatekopf des aufgerufenen Moduls. Weitere Informationen über die Registrierungsmakros, die *TRegList* generieren, finden Sie in den Einträgen der Registrierungsmakros in diesem Buch.

Diese beinhalten auch die Attribute, mit denen ein *TDocTemplate*-Objekt erzeugt wird.

Destruktor

```
~TDocTemplate();
```

Entfernt ein *TDocTemplate*-Objekt und gibt die Datenelemente (*FileFilter*, *Description*, *Directory* und *DefaultExt*) frei. Der Destruktor wird nur aufgerufen, wenn mit dem Template keine Views oder Dokumente assoziiert sind. Anstatt diesen Destruktor direkt aufzurufen, sollten Sie die Elementfunktion *Delete* verwenden.

Siehe auch: *dt xxxx* Dokument-Template-Konstanten

Klasse TDocTemplateT, <D,V> doctpl.h

Um die assoziierten Dokument- und View-Klassen zu registrieren, wird eine parametrisierte Subklasse mit Namen *TDocTemplateT*,<D,V> zum Konstruieren eines speziellen Dokumentes und View benutzt. Dabei repräsentiert *D* die Dokumentklasse und *V* die View-Klasse. Die parametrisierten Template-Klassen werden mit Hilfe eines Makros erzeugt, das zudem die assoziierte Unterstützung zur Streamfähigkeit generiert. Die Dokument- und View-Klassen werden unter Verwendung einer parametrisierten Subklasse bereitgestellt. Der Template-Klassenname wird als ein typedef für die parametrisierte Klasse benutzt. Ein Beispiel:

```
DEFINE_DOC_TEMPLATE_CLASS(TFileDocument, TEditView, MyEditFile)
```

Sie können ein Dokument-Template instantiieren, indem Sie entweder ein statisches Element oder eine explizite Konstruktion verwenden. Ein Beispiel:

```
MyEditFile et1("Edit text files",
               "*.txt","D:\\doc",".TXT"",dtNoAutoView);
new MyEditFile(.....)
```

Wenn ein neues Dokument-Template erzeugt wird, registriert die Dokumentverwaltung (*TDocManager*) das Template. Wenn die Löschfunktion des Dokument-Templates zum Löschen des Templates aufgerufen wird,

ist dieses Template für den Benutzer nicht mehr sichtbar. Es bleibt allerdings im Speicher, solange es von irgendeinem Dokument benutzt wird.

Öffentliche Konstruktoren

Konstruktor

Format 1

```
TDocTemplateT(const char far* filt, const char far* desc, const char
          far* dir, const char far* ext, long flags = 0, TModule*& module =
::Module,              TDocTemplate*& phead = DocTemplateStaticHead);
```

Konstruiert ein *TDocTemplateT* mit der angegebenen Dateibeschreibung (*desc*), der Dateifiltermaske (*filt*), dem Suchpfad für die Ansicht des Verzeichnisses (*dir*), der Standard-Dateinamenserweiterung (*ext*) und mit Flags (*flags*), die die Optionen zur Erzeugung und zur Ansicht repräsentieren. *module*, welches bereits direkt von jedem ausführbaren Modul instanziiert und exportiert wurde, kann dazu benutzt werden, auf die momentane Instanz Zugriff zu erhalten.

Format 2

```
TDocTemplateT(TRegList& regList, TModule*& module = ::Module,
  TDocTemplate*& phead = DocTemplateStaticHead);
```

Konstruiert ein *TDocTemplateT* und benutzt die Registriertabelle, um die Dateifiltermaske zu ermitteln, den Suchpfad für die Ansicht des Verzeichnisses, der Standard-Dateinamenserweiterung und der Flag-Werte. Im Eintrag über Registrierungsmakros finden Sie weitere Informationen darüber, wie Registriertabellen erstellt werden. *module*, welches bereits direkt von jedem ausführbaren Modul instanziiert und exportiert wurde, kann dazu benutzt werden, auf die momentane Instanz Zugriff zu erhalten.

Öffentliche Elementfunktionen

Clone

```
TDocTemplateT* Clone(TModule* module, TDocTemplate*& phead =
DocTemplateStaticHead;
```

Macht eine Kopie vom Objekt des Typs *TDocTemplateT*.

CreateDoc

```
D* CreateDoc(const char far* path, long flags = 0);
```

CreateDoc erzeugt ein Dokument vom Typ *D*, das auf dem Verzeichnispfad (*path*) und dem Wert *flags* basiert.

Siehe auch: *TDocTemplate::CreateDoc*

CreateView

```
TView* CreateView(TDocument& doc, long flags = 0);
```

CreateView erzeugt das View, das durch die Dokument-Template-Klasse angegeben ist.

Siehe auch: *TDocManager::CreateAnyView*

IsMyKindOfDoc

```
D* IsMyKindOfDoc(TDocument& doc);
```

IsMyKindOfDoc überprüft, ob das Dokument (*doc*) dieselbe Klasse ist wie die Dokumentklasse des Templates oder ob das Dokument eine abgeleitete Klasse ist. Wenn das Template das Dokument nicht benutzen kann, gibt *IsMyKindOfDoc* Null zurück.

Siehe auch: *TDocTemplate::IsMyKindOfDoc*

IsMyKindOfView

```
V* IsMyKindOfView(TView& view);
```

IsMyKindOfView überprüft, ob das View (*view*) dieselbe Klasse ist wie die View-Klasse des Templates oder ob das View eine abgeleitete Klasse ist. Wenn das Template das View nicht benutzen kann, gibt *IsMyKindOf-View* Null zurück.

GetViewName

```
virtual const char far* GetViewName();
```

GetViewName ermittelt den Namen des mit dem Template assoziierten View.

Klasse TDropInfo geometry.h

TDropInfo ist eine einfache Klasse, die Operationen zum Ziehen und Ablegen von Dateinamen mit Hilfe der Botschaft WM_DROPFILES unterstützt. Jedes *TDropInfo*-Objekt hat ein privates Handle auf die HDROP-Struktur, die durch die Botschaft WM_DOPFILES zurückgegeben wurde.

Öffentliche Konstruktoren

Konstruktor

```
TDropInfo(HDROP handle);
```

Erzeugt ein *TDropInfo*-Objekt, bei dem *Handle* auf das gegebene *handle* gesetzt ist.

Siehe auch: *TDropInfo::Handle*

Öffentliche Elementfunktionen

DragFinish

```
void DragFinish();
```

Gibt den Speicher frei, der für den Transfer von Dateien dieses *TDropInfo*-Objektes während der Operation zum Ziehen reserviert wurde.

DragQueryFile

```
UINT DragQueryFile( UINT index, char far* name, UINT nameLen);
```

Ermittelt den Dateinamen und bezogene Informationen für dieses *i*-Objekt. Wenn *index* auf -1 (0xFFFF) gesetzt ist, gibt *DragQueryFile* die Zahl der abgelegten Dateien zurück. Dies ist äquivalent zum Aufruf von *DragQueryFileCount*.

Wenn *index* zwischen Null und der Gesamtzahl abgelegter Dateien für dieses Objekt liegt, kopiert *DragQueryFile* den Namen der abgelegten Datei, die zu *index* korrespondiert, in den Puffer *name* (mit *nameLen* Bytes Länge) und gibt die Zahl der wirklich kopierten Bytes zurück.

Wenn *name* Null ist, gibt *DragQueryFile* die für den gegebenen *index* erforderliche Puffergröße (in Bytes) zurück. Dies ist äquivalent zum Aufruf von *DragQueryFileNameLen*.

Siehe auch: *TDropInfo::DragQueryPoint*, *TDropInfo::DragQueryFileCount*

DragQueryFileCount

```
UINT DragQueryFileCount();
```

Gibt die Zahl der in diesem *TDropInfo*-Objekt abgelegten Dateien zurück. Dieser Aufruf ist äquivalent zum Aufruf von *DragQueryFile(-1, 0, 0)*.

Siehe auch: *TDropInfo::DragQueryFile*

DragQueryFileNameLen `UINT DragQueryFileNameLen(UINT index)`

Gibt die Länge des Dateinamens in diesem *TDropInfo*-Objekt, der zum gegebenen Index korrespondiert, zurück. Dieser Aufruf ist äquivalent zum Aufruf von *DragQueryFile(index, 0, 0)*.

Siehe auch: *TDropInfo::DragQueryFile*

DragQueryPoint

```
BOOL DragQueryPoint(TPoint& point)
```

Ermittelt die Mauszeigerposition, wenn die Dateien dieses Objektes abgelegt werden, und kopiert die Koordinaten in das gegebene *point*-Objekt. *point* zeigt auf das Fenster, das die WM_DROPFILES-Botschaft empfangen hat. *DragQueryPoint* gibt true zurück, wenn das Ablegen innerhalb des Client-Bereiches des Fensters stattfindet. Anderenfalls gibt die Funktion false zurück.

Siehe auch: *::TPoint*, Klasse

HDROP()

```
operator HDROP();
```

Typumwandlungsoperator, der *Handle* zurückgibt.

Siehe auch: *TDropInfo::Handle*

Klasse TPoint — geometry.h

TPoint ist eine Unterstützungsklasse, die aus *tagPoint* abgeleitet ist. Unter Win32 ist sie wie folgt definiert:

```
typedef struct tagPoint {
      int x;
```

```
    int y;
} ;
```

In *TPoint* ist die Angabe eines Punktes in zwei Dimensionen eingekapselt, die normalerweise eine Bildschirmposition darstellt. *TPoint* erbt zwei Datenelemente von *tagPoint*, die Koordinaten *x* und *y*. Es stehen Elementfunktionen und Operatoren zum Vergleichen, Zuweisen und Manipulieren von Punkten zur Verfügung. Überladene **<<** und **>>** Operatoren werden als Friends von *TPoint* deklariert; so ist die verkettete Einfügung und Extraktion von *TPoint*-Objekten mit Streams möglich.

Öffentliche Konstruktoren

Konstruktor

Format 1

```
TPoint();
```

Der TPoint-Standardkonstruktor

Format 2

```
TPoint(int _x, int _y);
```

Erzeugt ein *TPoint*-Objekt mit den angegebenen Koordinaten.

Format 3

```
TPoint(const POINT point);
```

Erzeugt ein *TPoint*-Objekt mit $x = point.x$ und $y = point.y$.

Format 4

```
TPoint(const SIZE& size);
```

Erzeugt ein *TPoint*-Objekt mit $x = size.cx$ und $y = size.cy$

Format 5

```
TPoint(uint32 dw);
```

Erzeugt ein *TPoint*-Objekt mit $x = LOWORD(dw)$ und $y = HIWORD(dw)$.

Öffentliche Elementfunktionen

Offset

```
TPoint& Offset(int dx, int dy);
```

Versetzt den Punkt um die angegebenen Delta-Argumente. Dieser Punkt wird auf ($x + dx, y + dy$) versetzt. Vergibt eine Referenz auf diesen Punkt.

Siehe auch: *TPoint::OffsetBy, TPoint::operator+=*

OffsetBy

```
TPoint OffsetBy(int dx, int dy) const;
```

Berechnet einen Versatz (Offset) für diesen Punkt unter Verwendung der angegebenen Versatzargumente. Liefert den Punkt ($x + dx, y + dy$) zurück. Dieser Punkt wird nicht verändert.

Siehe auch: *TPoint::operator+, TPoint::Offset*

operator+

```
TPoint operator+(const TSize& size) const;
```

Errechnet einen Versatz (Offset) für diesen Punkt unter Verwendung des angegebenen *size*-Arguments für den Versatz. Gibt den Punkt (*x+size.cx, y+size.cy*) zurück. Dieser Punkt wird nicht verändert.

Siehe auch: *TPoint::OffsetBy, TSize*, Klasse

operator-

```
TPoint operator-(const TSize& size) const;
 TSize operator-(const TPoint& point) const;
 TPoint operator-() const;
```

Die erste Version berechnet einen negativen Versatz (Offset) für diesen Punkt und verwendet dafür das angegebene *size*-Argument für die Bemessung. Gibt den Punkt (*x - size.cx, y - size.cy*) zurück. Der Punkt wird nicht verändert.

Die zweite Fassung berechnet einen Abstand von diesem Punkt zum *point*-Argument. Gibt das *TSize*-Objekt zurück (*x - point.x, y - point.y*). Der Punkt wird nicht verändert.

Die dritte Fassung liefert den Punkt (*-x, -y*) zurück. Der Punkt wird nicht verändert.

Siehe auch: *TPoint::operator+, TSize*, Klasse

operator==

```
BOOL operator==(const TPoint& other) const;
```

Gibt true zurück, wenn dieser Punkt gleich dem Punkt *other* ist, andernfalls false.

Siehe auch: *TPoint::operator!=*

operator+=

```
TPoint& operator+=(const TSize& size);
```

Versetzt diesen Punkt um den Betrag des angegebenen *size*-Arguments. Der Punkt wird geändert in (*x + size.cx, y + size.cy*). Vergibt eine Referenz auf diesen Punkt.

Siehe auch: *TPoint::Offset, TPoint::operator-=, TSize*, Klasse

operator-=

```
TPoint& operator-=(const TSize& size);
```

Führt einen negativen Versatz (Offset) um den Betrag des angegebenen *size*-Arguments für diesen Punkt durch. Der Punkt wird in (*x - size.cx, y - size.cy*) geändert. Vergibt eine Referenz auf diesen Punkt.

Siehe auch: *TPoint::Offset, TPoint::*operator+=, *TSize*, Klasse

operator!=

```
BOOL operator!=(const TPoint& other) const;
```

Gibt false zurück, wenn dieser Punkt gleich dem Punkt *other* ist, andernfalls true.

Siehe auch: *TPoint::*operator==

operator>>

Format 1

```
ipstream& operator>>(ipstream& is, TPoint& p);
```

Extrahiert ein *TPoint*-Objekt aus *is*, dem angegebenen Eingabestream, und kopiert es nach *p*. Vergibt eine Referenz auf den resultierenden Stream und ermöglicht so die übliche Verkettung von **<<** Operationen.

operator<<

Format 2

```
istream& operator>>(istream& is, TPoint& p);
```

Extrahiert ein *TPoint*-Objekt aus *is*, dem angegebenen Eingabestream, und kopiert es nach *p*. Vergibt eine Referenz auf den resultierenden Stream und ermöglicht so die übliche Verkettung von **>>** Operationen.

Siehe auch: *TPoint* **<<** , *class ipstream*

operator<<

Format 1

```
opstream& operator<<(opstream& os, const TPoint& p);
```

Fügt das gegebene *TPoint*-Objekt *p* in den bestehenden stream *os* ein. Vergibt eine Referenz auf den resultierenden Stream und ermöglicht so die übliche Verkettung von **>>**-Operationen.

Format 2

```
opstream& operator<<(opstream& os, const TPoint& p);
```

Formatiert das gegebene *TPoint*-Objekt *p* und fügt es in den *ostream*, *os*, ein. Das Format ist »(x,y)«. Vergibt eine Referenz auf den resultierenden Stream und ermöglicht so die übliche Verkettung von **<<**-Operationen.

Siehe auch: *TPoint* **>y>** , *ostream*

Klasse TPointer<>> geometry.h

TPointer ist eine kleine Klasse von Dienstprogrammen und bietet die automatische Entfernung von Objekten, die unter Verwendung von **new** konstruiert wurden. *TPointer* ist eine parametrisierte Klasse, die einen Zeiger ihres parametrisierten Typs enthält und die Operatoren überlädt, so daß sie sich wie Objektzeiger verhalten. Ist einem *TPointer*-Objekt ein Zeiger zugewiesen, wird es beim Eintreten eines festgelegten Ereignisses gelöscht, wenn die Funktion existiert, wenn eine Exception ausgelöst wird oder wenn dem gleichen Objekt ein weiterer Zeiger zugewiesen wird. Ein *TPointer*-Objekt kann mit einer der folgenden gleichwertigen Methoden instantiiert werden:

```
TPointer<SomeClass> p = new SomeClass;
 TPointer<SomeClass> p(new SomeClass);
 TPointer<SomeClass> p;        p = new SomeClass;
```

Um ein Objekt zu entfernen, muß dem Zeiger 0 zugeweisen werden.

Öffentliche Konstruktoren

Konstruktor

Format 1

```
TPointer() : TPointerBase<T>();
```

Standardkonstruktor, in dem *p* mit 0 initialisiert ist.

Format 2

```
TPointer(T* pointer) : TPointerBase<T>(pointer);
```

Initialisierter Konstruktor, in dem *p* mit *pointer* initialisiert ist.

Öffentliche Elementfunktionen

operator *

```
T& operator *();
```

Überladener Operator für Typkonvertierungen, der seine Argumente in einen Zeiger auf den umgewandelten Typ castet.

operator T*

```
operator T*();
```

Überladener Operator für Typkonvertierungen, der es dem Objekt vom Typ *TPointer* erlaubt, als Funktionsargument umgewandelt zu werden oder so an eine Variable zugewiesen zu werden, wie wenn es sich um einen Zeiger handeln würde.

operator =

Format 1:

```
T* operator =(T* src);
```

Zuweisungsoperator *T* wird an *p* zugewiesen.

Format 2:

```
T* operator =(const TPointer<T>& src);
```

Der Zuweisungsoperator wird verwendet, wenn *r* eine **const** Referenz an *TPointer* vom Typ *T* ist. Dieser Operator sichert die Umwandlung, wenn ein anderes Zeigerobjekt verwendet wird.

operator ->

```
T* operator ->();
```

>>>Ermöglicht den Zugriff auf einen Zeiger.

Klasse TProcInstance geometry.h

TProcInstance wurde für Win16-Anwendungen entwickelt, und zwar für die Erzeugung und Freigabe eines Instanzen-Thunks, einem Codeabschnitt, der für die Verwendung exportierter Callback-Funktionen erzeugt wurde. (Eine Callback-Funktion ist eine Funktion, die zwar in

einem Programm vorkommt, jedoch von außerhalb des Programms mittels einer Windows Bibliotheksroutine, beispielsweise einer Dialogfenster-Routine, aufgerufen wird.) *TProcInstance* besitzt keine Funktionalität für Win32-Anwendungen. Die von *TProcInstance* zurückgelieferte Adresse kann als Parameter an Callback-Funktionen, Fenster-Unterklassen-Funktionen oder Windows Dialogfenster-Funktionen übergeben werden. Vergleiche die Windows-API-Hilfe mit weiteren Informationen zu *MakeProcInstance*, das einen Instanzen-Thunk erstellt, und *FreeProcInstance*, das einen Instanzen-Thunk freigibt.

Öffentliche Konstruktoren und Destruktor

Konstruktor

```
TProcInstance(FARPROC p);
```

Erzeugt eine *TProcInstance*. *p* wird als Adresse der Prozedur übergeben. Unter Win 16 wird *::MakeProcInstance* aufgerufen, um einen Instanzen-Thunk für *p* zu erstellen. Unter Win32 sichert der Konstruktor nur *p*.

Destruktor

```
~TProcInstance()
```

Gibt unter WIN16 den Instanzen-Thunk frei.

Siehe auch *::MakeProcInstance*, (Windows API), *::FreeProcInstance* (Windows API)

Öffentliche Elementfunktionen

FARPROC
operator FARPROC

```
OPERATOR FARPROC();
```

Unter Win 16 wird der Instanzen-Thunk zurückgegeben. Unter Win32 wird *p* vom Konstruktor zurückgegeben.

Klasse TRect — geometry.h

TRect ist eine von *tagRect* abgeleitete unterstützende Klasse. Unter Win32 ist *tagRect* definiert als

```
typedef struct tagRECT {
     int left;
     int top;
     int right;
     int bottom;
} ;
```

TRect kapselt die Eigenschaften von Rechtecken mit zu den x- und y-Achsen parallelen Seiten. In ObjectWindows bestimmen diese Rechtecke die

Begrenzungen von Fenstern, Rahmen und Clipping-Bereichen. *TRect* erbt vier Datenelemente von *TagRect*: *left*, *top*, *right* und *bottom*. Diese stehen für die Koordinaten der oberen linken und unteren rechten Ecke (*x*, *y*) des Rechtecks. Beachten Sie, daß *x* von links nach rechts zunimmt, *y* von oben nach unten.

TRect macht keine Einschränkungen bezüglich der relativen Positionen von links oben und rechts unten, so daß *left* > *right* und *top* > *bottom* zulässig sind. Viele Manipulationen jedoch - wie die Bestimmung von Breite und Höhe und die Herstellung von Verbindungen und Schnittpunkten - lassen sich durch die *Normalisierung* der beteiligten *TRect*-Objekte vereinfachen. Ein Rechteck normalisieren bedeutet, die Koordinaten seiner Eckpunkte zu vertauschen, so daß *left* < *right* und *top* < bottom. Die Normalisierung ändert nicht die physischen Eigenschaften eines Rechtecks. *myRect.Normalized* erzeugt eine normalisierte Kopie von *myRect*, ohne *myRect* zu verändern, während *myRect.Normalize* das Format von *myRect* durch Normalisieren verändert. Beide Elemente geben das normalisierte Rechteck zurück.

TRect-Konstruktoren stehen zur Verfügung, um Rechtecke aus entweder vier **int** s, zwei *TPoint*-Objekten oder einem *TPoint*- und einem *TSize*-Objekt zu erzeugen. Im letzten Fall bestimmt das *TPoint*-Objekt den oberen linken Eckpunkt (auch als Ursprung des Rechtecks bezeichnet), während das *TSize*-Objekt Höhe und Breite des Rechtecks angibt. Elementfunktionen führen eine Vielzahl von Tests und Manipulationen mit Rechtecken durch. Überladene << und >> -Operatoren werden als Friends von *TRect* deklariert und erlauben so das verkettete Einfügen und Entfernen von *TRect*-Objekten mit Streams.

Öffentliche Konstruktoren

Format 1

```
TRect();
```

Der voreingestellte Konstruktor.

Format 2

```
TRect(const RECT far& rect);
```

Kopiert das vorliegende *rect* in dieses Objekt.

Format 3

```
TRect(int _left, int _top, int _right, int _bottom);
```

Erzeugt ein Rechteck mit den gegebenen Werten.

Format 4

```
TRect(const TPoint& upLeft, const TPoint& loRight);
```

Erzeugt ein Rechteck mit den gegebenen oberen linken und unteren rechten Eckpunkten.

Siehe auch: *TPoint*

Format 5

```
TRect(const TPoint& origin, const TSize& extent);
```

Erzeugt ein Rechteck, das seinen Ursprung (oben links) in *origin* hat, sich in der Breite bis zu *extent.cx* und in der Höhe bis zu *extent.cy* erstreckt.

Siehe auch: *TPoint*, *TSize*

Öffentliche Elementfunktionen

Area

```
long Area() const;
```

Gibt den Bereich des vorliegenden Rechtecks zurück.

Siehe auch: *TRect::Size*

BottomLeft

```
TPoint BottomLeft() const;
```

Gibt das *TPoint*-Objekt zurück, welches die obere linke Ecke des vorliegenden Rechtecks repräsentiert.

Siehe auch: *TRect::TopLeft*, *TRect::TopRight*, *TRect::BottomRight*

BottomRight

```
const TPoint& BottomRight() const;
TPoint& BottomRight();
```

Gibt das *TPoint*-Objekt zurück, welches die untere rechte Ecke des vorliegenden Rechtecks repräsentiert.

Siehe auch: *TRect::TopRight*, *TRect::BottomLeft*, *TRect::TopLeft*

Contains

```
BOOL Contains(const TPoint& point) const;
BOOL Contains(const TRect& other) const;
```

Die erste Version liefert true zurück, wenn der gegebene *point* innerhalb des Rechtecks liegt; andernfalls wird false zurückgeliefert. Liegt *point* auf dem linken vertikalen oder oberen horizontalen Rand des Rechtecks, liefert *Contains* ebenfalls true zurück, liegt *point* jedoch auf dem rechten vertikalen oder unteren horizontalen Rand, liefert *Contains* false zurück.

Die zweite Version liefert true zurück, wenn das andere Rechteck auf oder in dem vorliegenden Rechteck liegt; andernfalls wird false zurückgeliefert.

Siehe auch: *TRect::Touches*, *TPoint*, *TRect*

Height

```
int Height() const;
```

Gibt die Höhe des vorliegenden Rechtecks zurück (*bottom* - *top*).

Siehe auch: *TRect::Width*

Inflate

```
TRect& Inflate(int dx, int dy);
TRect& Inflate(const TSize& delta);
```

Ändert die Größe eines Rechtecks mittels des mitgegebenen Argumentes Delta. In der ersten Version ist die obere linke Seite des zurückgegebenen Rechtecks (*left - dx*, *top - dy*), während seine untere rechte Seite (*right + dx*, *bottom + dy*) ist. In der zweiten Version sind die neuen Ecken (*left - size.cx*, *top - size.cy*) und (*right + size.cx*, *bottom + size.cy*).

Siehe auch: *TRect*, Klasse, *TSize*, Klasse

InflatedBy

```
TRect InflatedBy(int dx, int dy) const;
 TRect InflatedBy(const TSize& size) const;
```

Gibt ein durch die gegebenen Delta-Argumente vergrößertes Rechteck zurück. In der ersten Version ist die obere linke Ecke des zurückgelieferten Rechtecks (*left - dx*, *top - dy*), während die untere rechte Ecke (*right + dx*, *bottom + dy*) ist. In der zweiten Version sind die neuen Eckpunkte (*left - size.cx*, *top - size.cy*) und (*right + size.cx*, *bottom + size.cy*). Das aufrufende Rechteckobjekt bleibt unverändert.

Siehe auch: *TRect::OffsetBy*, *Trect*, *TSize*

IsEmpty

```
BOOL IsEmpty() const;
```

Gibt true zurück, wenn *left* >= *right* oder *top* >= *bottom*; andernfalls wird false zurückgeliefert.

Siehe auch: *TRect::SetEmpty*, *TRect::IsNull*

IsNull

```
BOOL IsNull() const;
```

Gibt true zurück, wenn sowohl *left* als auch *right*, *top* und *bottom* 0 sind; andernfalls wird false zurückgeliefert.

Siehe auch: *TRect::IsEmpty*, *TRect::SetEmpty*

Normalize

```
TRect& Normalize();
```

Normalisiert das vorliegende Rechteck, indem die Werte der *left*- und *right*-Datenelemente umgeschaltet werden, wenn *left* > *right*, und die Werte der *top*- und *bottom*-Datenelemente umgeschaltet werden, wenn *top* > *bottom*. *Normalize* gibt ein normales Rechteck zurück. Ein gültiges, aber nicht-normales Rechteck könnte *left* > *right* und/oder *top* > *bottom* aufweisen. Dies macht viele Manipulationen (etwa die Festlegung von Breite und Höhe) unnötig kompliziert. Ein Rechteck normalisieren bedeutet, die Werte der Eckpunkte zu vertauschen, so daß *left* < *right* und *top* < *bottom*. Die physischen Eigenschaften eines Rechtecks bleiben bei diesem Prozeß unverändert.

Siehe auch: *TRect::Normalized*

Normalized

```
TRect Normalize() const;
```

Gibt ein normalisiertes Rechteck mit der oberen linken Ecke in (Min(*left*, *right*), Min(*top*, *bottom*)) und der unteren rechten Ecke in (Max(*left*,

right), Max(*top*, *bottom*)) zurück. Das aufrufende Rechteckobjekt bleibt unverändert. Ein gültiges, aber nicht-normales Rechteck könnte *left* > *right* und/oder *top* > *bottom* aufweisen. Dies macht viele Manipulationen (etwa die Festlegung von Breite und Höhe) unnötig kompliziert. Ein Rechteck normalisieren bedeutet, die Werte der Eckpunkte zu vertauschen, so daß *left* < *right* und *top* < *bottom*. Die physischen Eigenschaften eines Rechtecks bleiben bei diesem Prozeß unverändert.

Beachten Sie, daß viele Rechenoperationen ein normalisiertes Rechteck voraussetzen. Einige Windows-API-Funktionen zeigen fehlerhaftes Verhalten, wenn ein umgestülptes *Rect* übergeben wird.

Siehe auch: *TRect::Normalize*, *TRect*

Offset

```
TRect& Offset(int dx, int dy);
```

Verändert das vorliegende Rechteck in der Weise, daß seine Ecken durch die gegebenen Delta-Werte versetzt werden. Das überarbeitete Rechteck hat seine obere linke Ecke in (*left* + *dx*, *top* + *dy*) und seine untere rechte Ecke in (*right* + *dx*, *bottom* + *dy*). Zurückgeliefert wird das geänderte Rechteck.

Siehe auch: *TRect::operator+*, *TRect::operator+=*, *TRect::OffsetBy*

OffsetBy

```
TRect OffsetBy(int dx, int dy) const;
```

Liefert ein Rechteck zurück, dessen Ecken durch die gegebenen Delta-Werte versetzt wurden. Das zurückgelieferte Rechteck hat seine obere linke Ecke in (*left* + *dx*, *top* + *dy*) und seine untere rechte Ecke in (*right* + *dx*, *bottom* + *dy*).

Siehe auch: *TRect::operator+*

operator+

```
TRect operator+(const TSize& size) const;
```

Liefert ein Rechteck zurück, das durch die gegebenen *size*-Delta-Werte positiv versetzt wurde. Das zurückgelieferte Rechteck hat seine obere linke Ecke in (*left* + *size.x*, *top* + *size.y*) und seine untere rechte Ecke in (*right* + *size.x*, *bottom* + *size.y*). Das aufrufende Rechteckobjekt bleibt unverändert.

Siehe auch: *TRect::OffsetBy*, *TSize*

operator-

```
TRect operator-(const TSize& size) const;
```

Liefert ein Rechteck zurück, das durch die gegebenen *size*-Delta-Werte negativ versetzt wurde. Das zurückgelieferte Rechteck hat seine obere linke Ecke in (*left* - *size.cx*, *top* - *size.cy*) und seine untere rechte Ecke in (*right* - *size.x*, *bottom* - *size.cy*). Das aufrufende Rechteckobjekt bleibt unverändert.

Siehe auch: *TRect::*offsetBy, *TSize*

operator&

```
TRect operator&(const TRect& other) const;
```

Liefert die Schnittmenge dieses und des Rechtecks in *other* zurück. Das aufrufende Rechteckobjekt bleibt unverändert. Schneiden sich die beiden Rechtecke nicht, wird ein NULL-Rechteck zurückgeliefert.

Siehe auch: *TRect::operator|*, *TRect::operator+=*

operator|

```
TRect operator|(const TRect& other) const;
```

Liefert die Vereinigungsmenge dieses und des Rechtecks in *other zurück.* Das aufrufende Rechteckobjekt bleibt unverändert.

Siehe auch: *TRect::operator&*, *TRect::operator|=*

operator==

```
BOOL operator==(const TRect& other) const;
```

Liefert true zurück, wenn das vorliegende Rechteck dieselben Eckpunktkoordinaten hat wie das Rechteck in *other*; andernfalls wird FALSE zurückgeliefert.

Siehe auch: *TRect::operator!=*

operator!=

```
BOOL operator!=(const TRect& other) const;
```

Liefert FALSE zurück, wenn das vorliegende Rechteck dieselben Eckpunktkoordinaten hat wie das Rechteck in *other*; andernfalls wird true zurückgeliefert.

Siehe auch: *TRect::operator==*

operator+=

```
TRect& operator+=(const TSize& delta);
```

Ändert das vorliegende Rechteck in der Weise, daß seine Eckpunkte durch die gegebenen Delta-Werte *delta.x* und *delta.y* versetzt werden. Das überarbeitete Rechteck hat seine obere linke Ecke in (*left* + *delta.x*, *top* + *delta.y*), seine untere rechte Ecke in (*right* + *delta.x*, *bottom* + *delta.y*). Zurückgeliefert wird das geänderte Rechteck.

Siehe auch: *TRect::operator+*, *TRect::OffsetBy*, *TRect::Offset*

operator-=

```
TRect& operator-=(const TSize& delta);
```

Ändert das vorliegende Rechteck in der Weise, daß seine Eckpunkte durch die gegebenen Delta-Werte *delta.x* und *delta.y* negativ versetzt werden. Das überarbeitete Rechteck hat seine obere linke Ecke in (*left* - *delta.x*, *top* - *delta.y*), seine untere rechte Ecke in (*right* - *delta.x*, *bottom* - *delta.y*). Zurückgeliefert wird das geänderte Rechteck.

Siehe auch: *TRect::operator-*, *TRect::operator+=*, *TRect::OffsetBy*, *TRect::Offset*@MARGINAL = operator&=@PROGTEXT = TRect& operat&=(const TRect& other) const;

Verändert das vorliegende Rechteck auf die Größe der Schnittmenge mit dem Rechteck in *other*. Dieses Rechteckobjekt wird zurückgeliefert. Gibt es keine Schnittmenge, so wird ein NULL-Rechteck zurückgeliefert.

Siehe auch: *TRect::operator&*, *TRect::operator|=*

operator|=

```
TRect& operator|=(const TRect& other) const;
```

Verändert das vorliegende Rechteck auf die Größe der Vereinigungsmenge mit dem Rechteck in *other*. Dieses Rechteckobjekt wird zurückgeliefert.

Siehe auch: *TRect::operator|*, *TRect::operator&=*

operator >>

```
ipstream& _BIDSFUNC operator >>(ipstream& is, TRect& r);
```

>> Nimmt aus einem gegebenen Eingabe-Stream *is* ein Objekt vom Typ *TRect* und kopiert es nach *r*. Gibt eine Referenz auf den Ergebnis-Stream zurück und ermöglicht die übliche Verkettung von Operationen mit >>.

Siehe auch: Operator *TRect* <<

operator <<

Deklaration 1:

```
opstream& _BIDSFUNC operator <<(opstream& os, const TRect& r);
```

Fügt das gegebene Objekt *r* vom Typ *TRect* in den *opstream os* ein. Gibt eine Referenz auf den Ergebnis-Stream zurück und ermöglicht damit die übliche Verkettung von Operationen mit <<.

Deklaration 2:

```
ostream& _BIDSFUNC operator <<(ostream& os, const TRect& r);
```

Formatiert das gegebene Objekt *r* vom Typ *TRect* und fügt es in der *ostream os* ein. Das Format ist dabei (*r.left, r.top*)(*r.right, r.bottom*). Gibt eine Referenz auf den Ergebnis-Stream zurück und ermöglicht damit die übliche Verkettung von Operationen mit <<.

Siehe auch: Operator *TRect* >>

operator TPoint*()

```
operator const TPoint*() const;
 operator TPoint*();
```

Typumwandlungsoperatoren, die den Zeiger auf das vorliegende Rechteck in den Typ Zeiger auf *TPoint* umwandeln.

Siehe auch: *TPoint*, Klasse

Set

```
void Set(int _left, int _top, int _right, int _bottom);
```

Verändert Position und Größe des vorliegenden Rechtecks gemäß den gegebenen Werten.

SetEmpty

```
void SetEmpty();
```

Leert das vorliegende Rechteck, indem *left*, *top*, *right* und *bottom* auf 0 gesetzt werden.

SetNull

```
void SetNull();
```

Setzt *left*, *top*, *right* und *bottom* eines Rechtecks auf 0.

Size

```
TSize Size() const;
```

Liefert ein *TSize*-Objekt zurück, das die Breite und Höhe des vorliegenden Rechtecks repräsentiert.

Siehe auch: *TSize*

TopLeft

```
const TPoint& TopLeft() const;
 TPoint& TopLeft();
```

Liefert das *TPoint*-Objekt zurück, welches die obere linke Ecke des vorliegenden Rechtecks repräsentiert.

Siehe auch: *TRect::TopRight*, *TRect::BottomLeft*, *TRect::BottomRight*

TopRight

```
TPoint TopRight() const;
```

Liefert das *TPoint*-Objekt zurück, welches die obere rechte Ecke des vorliegenden Rechtecks repräsentiert.

Siehe auch: *TRect::TopLeft*, *TRect::BottomLeft*, *TRect::BottomRight*

Touches

```
BOOL Touches(const TRect& other) const;
```

Liefert true zurück, wenn das Rechteck in *other* irgendeinen inneren Punkt mit dem vorliegenden Rechteck gemeinsam hat; andernfalls wird false zurückgeliefert.

Siehe auch: *TRect::Contains*

Width

```
int Width() const;
```

Liefert die Breite des vorliegenden Rechtecks zurück (*right* - *left*).

Siehe auch: *TRect::Height*

Klasse TResId — geometry.h

TResId ist eine einfache Supportklasse, die ein Ressourcen-ID-Objekt aus entweder einem numerischen oder einem String-Ressourcenbezeichner erstellt. Zum Beispiel kapselt *TResId* den Gebrauch von LPSTR(**char_far***) als ein Ressourcen-ID. Dieses Ressourcen-ID-Objekt kann an verschiedene ObjectWindows-Klassen übergeben werden. Um diese beiden verschiedenen Typen von Rsourcen-ID handhaben zu können,definiert *TResId* einen Konversionsoperator und verschafft sich zwei Konstruktoren, die diese dann konvertieren und als einfache Datentypen behandeln. Der

eine Konstruktor akzeptiert eine 16-Bit-Ganzzahl und der andere eine Zeichenkette.

Öffentliche Konstruktoren

Konstruktoren

Format 1

```
TResID();
```

Der voreingestellte *TResID*-Konstruktor.

Format 2

```
TResID(int resNum);
```

Erzeugt ein *Objekt TResID* mit der gegebenen *resNum*.

Format 3

```
TResID(const char far* resString);
```

Erzeugt mit dem gegebenen *resString* ein *ResID*-Objekt.

Öffentliche Elementfunktionen

char far*

```
operator char far*();
```

Typcasting-Operator, der *Id* (ein *private* Datenelement von *TResId*) in den Typ **char far*** umwandelt, so daß die Instanzen von *TResId* überall dort verwendet werden können, wo der Datentyp **char far*** erwartet wird.

IsString

```
BOOL IsString() const;
```

Liefert true zurück, wenn diese Ressourcen-ID aus einem String erstellt wurde; andernfalls wird false zurückgeliefert.

Friend-Funktionen

>>>>>>;TResIDFriendOperator>>

```
friend ipstream& operator>>(ipstream& is, TResID& id);
```

Zieht ein Objekt *TResID* aus *is* (dem gegebenen Eingabestream) und kopiert es nach *id*. Liefert, unter Zulassung der üblichen Verkettung von **>>** -Operationen, eine Referenz zum Ergebnisstream zurück.

Siehe auch: *TResID friend operator* **<<** *, ipstream*

Operator<<

Format 1

```
friend opstream& operator<<(opstream& os, const TResID& id);
```

Fügt das gegebene Objekt *TResID* (*id*) in den *opstream* (*os*) ein. Liefert eine Referenz zum Ergebnisstream unter Zulassung der üblichen Verkettung von **<<** Operationen zurück.

Format 2

```
friend ostream& operator <<(ostream& os, const TResID& id);
```

Formatiert das gegebene Objekt *TResID* (*id*) und fügt es in den *ostream* (*os*) ein. Liefert eine Referenz zum Ergebnisstream unter Zulassung der üblichen Verkettung von **<<** -Operationen zurück.

Siehe auch: *TResID friend operator*>>, *ostream, opstream*

Klasse TSize geometry.h

TSize ist eine aus der Struktur *tagSIZE* abgeleitete mathematische Klasse. Die Struktur *tagSize* wird wiefolgt definiert

```
struct tagSIZE {
    int cx;
    int cy;
} ;
```

TSize steht zusammenfassend für den Begriff einer zweidimensionalen Menge, die normalerweise ein Versetzen der Höhe und Breite eines Rechtecks bedeutet. *TSize* erbt die beiden Datenelemente *cx* und *cy* von *tagSIZE*. Es stehen Elementfunktionen und Operatoren zum Vergleichen, Zuweisen und Manipulieren von Größen zur Verfügung. Überladene **<**- und **>** Operatoren erlauben das verkettete Einfügen und Entfernen von *TSize*-Objekten mit Streams.

Öffentliche Konstruktoren

Format 1

```
TSize();
```

Der standardmäßige *TSize*-Konstruktor.

Format 2

```
TSize(int dx, int dy);
```

Erzeugt ein Objekt *TSize* mit *cx* = *dx* und *cy* = *dy*.

Format 3

```
TSize(const POINT& point);
```

Erzeugt ein Objekt *TSize* mit *cx* = *point.x* und *cy* = *point.y*.

Format 4

```
TSize(const SIZE& size);
```

Erzeugt ein Objekt *TSize* mit *cx* = *size.cx* and *cy* = *size.cy*.

Format 5

```
TSize(UINT32 dw);
```

Erzeugt ein Objekt *TSize* mit *cx* = *LOWORD*(*dw*) und *cy* = *HIWORD*(*dw*).

Siehe auch: *TPoint, Size*

Öffentliche Elementfunktionen

Magnitude

```
int Magnitude() const;
```

Liefert die Länge der Diagonale des durch dieses Objekt repräsentierten Rechtecks zurück. Der zurückgelieferte Wert ist eine **int** -Annäherung an die Quadratwurzel aus ($cx^2 + cy^2$).

operator+

```
TSize operator+(const TSize& size) const;
```

Berechnet eine Versetzung dieses Objektes *TSize* unter Verwendung des gegebenen *size*-Arguments als Faktor für die Verschiebung. Liefert das Objekt (*cx* + *size.cx*, *cy* + *size.cy*) zurück. Das vorliegende Objekt *TSize* bleibt unverändert.

Siehe auch: *TSize::operator-*

operator-

```
TSize operator-(const TSize& size) const;
 TSize operator-() const;
```

Die erste Version berechnet eine negative Versetzung des vorliegenden Objektes *TSize* unter Verwendung des gegebenen *size*-Arguments als Faktor für die Verschiebung. Liefert den Punkt (*cx* - *size.cx*, *cy* - *size.cy*) zurück. Dieses Objekt bleibt unverändert.

Die zweite Version liefert das Objekt *TSize* (- *cx*, - *cy*) zurück. Dieses Objekt wird nicht geändert.

Siehe auch: *TSize::operator+*

operator==

```
BOOL operator==(const TSize& other) const;
```

Liefert true zurück, wenn das vorliegende Size-Objekt gleich dem *TSize*-Objekt in *other* ist; ansonsten wird false zurückgeliefert.

Siehe auch: *TSize::operator!=*

operator!=

```
BOOL operator!=(const TSize& other) const;
```

Liefert true zurück, wenn das vorliegende Size-Objekt gleich dem *TSize*-Objekt in *other* ist; ansonsten wird false zurückgeliefert.

Siehe auch: *TSize::operator==*

operator+=

```
TSize& operator+=(const TSize& size) const;
```

Versetzt dieses *TSize*-Objekt mit Hilfe des gegebenen *size*-Arguments. Das vorliegende *TSize*-Objekt wird geändert zu (*cx* + *size.cx*, *cy* + *size.cy*). Es wird eine Referenz zum vorliegenden Objekt zurückgeliefert.

Siehe auch: *TSize::operator-=*

operator-=

```
TSize& operator-=(const TSize& size) const;
```

Versetzt dieses *TSize*-Objekt mit Hilfe des gegebenen *size*-Arguments in negativer Weise. Das vorliegende *TSize*-Objekt wird geändert zu (*cx - size.cx*, *cy - size.cy*). Es wird eine Referenz zum vorliegenden Objekt zurückgeliefert.

Siehe auch: *TSize::operator-=*

operator >>>>;TSize-Friend

```
ipstream& operator>>(ipstream& is, TSize& s) const;
```

Entfernt ein *TSize*-Objekt aus *is*, dem gegebenen Eingabestream, und kopiert es nach *s*. Liefert eine Referenz zum Ergebnisstream unter Zulassung der üblichen Verkettung von **>>** -Operationen zurück.

Siehe auch: *TSize* operator **<<**, *ipstream*

operator<<

Format 1

```
opstream& operator<<(opstream& os, const TSize& s) const;
```

Fügt das gegebene *TSize*-Objekt (*s*) in den *opstream* (*os*) ein. Liefert eine Referenz zum Ergebnisstream unter Zulassung der üblichen Verkettung von **<<** -Operationen zurück.

Format 2

```
ostream& operator<<(ostream& os, const TSize& s) const;
```

Formatiert das gegebene *TSize*-Objekt (*s*) und fügt es in den *ostream* (*os*) ein. Das Format ist "(*cx* x *cy*)". Liefert eine Referenz zum Ergebnisstream unter Zulassung der üblichen Verkettung von **<<** Operationen zurück.

Siehe auch: *TSize* operator**>>,** *opstream, ostream*

Klasse TXBase except.h

TXBase ist von *xmsg* abgeleitet und ist die Basisklasse für die Exception-Behandlungsklassen für ObjectWindows und ObjectConnections. Die Klassen von ObjectWindows, die bestimmte Arten von Exceptions wie z.B. Speicherüberlauf oder Fensterfehler behandeln, sind von *TXOwl* abgeleitet, das wiederum von *TXBase* abgeleitet wurde. Die Klassen von *TXOle* und *TXAuto* von ObjectConnection sind direkt von *TXBase* abgeleitet.

TXBase enthält die Funktionen *Clone* and *Throw*, die in allen abgeleiteten Klassen überschrieben wurden, auch in den beiden Konstruktoren. Die Konstruktoren inkrementieren *InstanceCount*, das öffentliche Datenelement von *TXBase*, und der Destruktor dekrementiert *InstanceCount* wieder.

Im Borland C++ Referenzhandbuch befindet sich eine Beschreibung von *xmsg*, der übergeordneten Klasse von *TXBase*. Im ObjectWindows Pro-

grammierhandbuch finden Sie Informationen über die Verwendung von *TXBase* in Ihren Applikationen.

Siehe auch
TXOwl

Öffentliche Konstruktoren und Destruktoren

Konstruktoren

Format 1:

```
TXBase(const string& msg);
```

Ruft den Konstruktor der Klasse *xmsg* mit einem Stringparameter auf und initialisiert *xmsg* mit dem Wert des Stringparameters.

Format 2:

```
TXBase(const TXBase& src);
```

Erzeugt eine Kopie des Objekts *TXBase*, welches im Parameter *TXBase* übergeben wird.

Destructor

```
virtual ~TXBase;
```

Zerstört das *TXBase-Objekt* und dekrementiert das Datenelement *InstanceCount.*

Siehe auch: TXOwl Öffentliche Konstruktoren und Destruktoren

Öffentliche Datenelemente

InstanceCount

```
static int InstanceCount;
```

Zählt die Anzahl der Objekte der Klasse *TXBase* und der davon abgeleiteten Klassen, die in einer einzigen Applikation existieren.

Öffentliche Elementfunktionen

Clone

```
virtual TXBase* Clone();
```

Kopiert das Exception-Objekt.

Throw

```
virtual void Throw();
```

Wirft ein Exception-Objekt aus.

A N H A N G A

Gekapselte WIN-API-Funktionen

Dieses Kapitel enthält mehrere Tabellen, die gekapselte Windows-API-Funktionen, welche ein HWND als erstes Argument erwarten, auflisten. Die Tabellen sind auf folgende Art organisiert:

Tabelle	Beschreibung	Seite
4.1	Gekapselte-HWND-Funktionen	480
4.2	Gekapselte Windows-Botschaften	480
4.3	Fensterkoordinaten- und -größen	480
4.4	Fenstereigenschaften und -attribute	481
4.5	Fensterplazierung und -anzeige	481
4.6	Fensterverhältnisse	482
4.7	Fensterzeichenfunktionen	482
4.8	Funktionen zum Blättern von Fenstern	482
4.9	Funktionen mit der ID untergeordneter Fenster	483
4.10	Menü- und Menüleistenfunktionen	483
4.11	Zwischenablagefunktionen	484
4.12	Timer-Funktionen	484
4.13	Cursor- und Mauszeigerfunktionen	484
4.14	Tastenkürzelfunktionen	484
4.15	Hilfs- und Task-Funktionen	485

Die meisten gekapselten Funktionen sind als *inline*-Funktionen implementiert, die die Elementvariable *HWindow* als das HWND-Argument übergeben. Die übrigen Argumente werden ohne Änderung ihres Prototyps übergeben.

Die anderen Funktionen sind statische Funktionen, die Informationen hinsichtlich der Fenster zurückgeben. Diese Funktionen verwenden nicht tatsächlich *HWindow* als das Argument, da das HWND entweder mit einbegriffen ist oder die Funktion das HWND zurückgibt. Ein HWND, das als Handle auf ein ObjectWindows-Fenster dient, kann über *GetWindowPtr()* auf ein *TWindow** konvertiert werden.

Beachten Sie, daß im Gültigkeitsbereich von *TWindow* oder abgeleiteten Klassen alle direkten Aufrufe der korrespondierenden Windows-Versionen dieser Funktionen in einem globalen Gültigkeitsbereich liegen müssen, wie in *::SendMessage()*

Die Funktionen in Tabelle A.1 ermöglichen es, daß ein *TWindow* in Windows-API-Aufrufen wie ein HWND verwendet werden kann.

Tabelle A.1: Gekapselte *inline*-HWND-Funktionen

Name	Funktionsdeklaration
HWND	operator HWND () const
IsWindow	BOOL IsWindow () const

Die Funktionen in Tabelle A.2 behandeln Windows-Botschaften.

Tabelle A.2.: Gekapselte Windows-Botschaften

Name	Funktionsdeklaration
EnableWindow	BOOL EnableWindow (BOOL enable)
GetCapture	static HWND GetCapture ()
GetFocus	static HWND GetFocus ()
IsWindowEnabled	BOOL IsWindowEnabled () const
PostMessage	BOOL PostMessage (UINT msg, WPARAM wParam = 0, LPARAM lParam = 0) const
ReleaseCapture	static void ReleaseCapture ()
SendDlgItemMessage	LRESULT SendDlgItemMessage (int childId, UINT msg, WPARAM wParam = 0, LPARAM lParam = 0) const
SendMessage	LRESULT SendMessage (UINT msg, WPARAM wParam = 0, LPARAM lParam = 0) const
SetCapture	HWND SetCapture ()
SetFocus	HWND SetFocus ()

Tabelle A.3 listet Funktionen auf, die Fensterkoordinaten und -größen anpassen.

Tabelle A.3: Fensterkoordinaten und -größen

Name	Funktionsdeklaration
AdjustWindowRect	static void AdjustWindowRect (TRect& rect, DWORD style, BOOL menu)
AdjustWindowRectEx	static void AdjustWindowRectEx (TRect& rect, DWORD style, BOOL menu, DWORD exStyle)
ChildWindowFromPoint	HWND ChildWindowFromPoint (const TPoint& point)
ClientToScreen	void ClientToScreen (TPoint& point) const
GetClientRect	TRect GetClientRect ()
GetClientRect	void GetClientRect (TRect& rect)
GetWindowRect	void GetWindowRect (TRect& rect)
MapWindowPoints	void MapWindowPoints (HWND hWndTo, TPoint *points, int count) const

ScreenToClient	void ScreenToClient (TPoint& point) const
WindowFromPoint	static HWND WindowFromPoint (const TPoint& point)

Tabelle A.4 listet Funktionen auf, die Fenstereigenschaften und -attribute kapseln.

Tabelle A.4: Fenstereigenschaften und -attribute

Name	Funktionsdeklaration
EnumProps	int EnumProps (PROPENUMPROC proc)
GetClassLong	long GetClassLong (int index) const
GetClassName	long GetClassName (char far* className, int maxCount) const
GetClassWord	WORD GetClassWord (int index) const
GetProp	HANDLE GetProp (const char far* str) const
GetProp	HANDLE GetProp (WORD atom) const
GetWindowLong	long GetWindowLong (int index) const
GetWindowWord	WORD GetWindowWord (int index) const
RemoveProp	HANDLE RemoveProp (const char far* str) const
RemoveProp	HANDLE RemoveProp (WORD atom) const
SetClassLong	long SetClassLong (int index, long newLong)
SetClassWord	WORD SetClassWord (int index, WORD newWord)
SetProp	BOOL SetProp (const char far* str, HANDLE data) const
SetProp	BOOL SetProp (WORD atom, HANDLE data) const
SetWindowLong	long SetWindowLong (int index, long newLong)
SetWindowWord	WORD SetWindowWord (int index, WORD newWord)

Tabelle A.5 listet Funktionen auf, die Eigenschaften zur Fensterplazierung und -anzeige kapseln.

Tabelle A.5: Fensterplazierung- und -anzeige

Name	Funktionsdeklaration
GetWindowPlacement	BOOL GetWindowPlacement (WINDOWPLACEMENT *place) const
GetWindowText	int GetWindowText (char far* str, int maxCount) const
GetWindowTextLength	int GetWindowTextLength () const
IsIconic	BOOL IsIconic () const
IsWindowVisible	BOOL IsWindowVisible () const
IsZoomed	BOOL IsZoomed () const
MoveWindow	void MoveWindow (const TRect& rect, BOOL repaint = FALSE)
MoveWindow	void MoveWindow (int x, int y, int w, int h, BOOL repaint = FALSE)
SetWindowPlacement	BOOL SetWindowPlacement (const WINDOWPLACEMENT *place)
SetWindowText	void SetWindowText (const char far* str)
ShowOwnedPopups	void ShowOwnedPopups (BOOL show)
ShowWindow	BOOL ShowWindow (int cmdShow)

Die Funktionen in Tabelle A.6 kontrollieren Fensterpositionen und gleichgeordnete Verhältnisse.

Tabelle A.6: Fensterverhältnisse

Name	Funktionsdeklaration
BringWindowToTop	void BringWindowToTop ()
GetActiveWindow	static HWND GetActiveWindow ()
GetDesktopWindow	static HWND GetDesktopWindow ()
GetLastActivePopup	HWND GetLastActivePopup () const
GetNextWindow	HWND GetNextWindow (UINT dirFlag) const
GetSysModalWindow	static HWND GetSysModalWindow ()
GetTopWindow	HWND GetTopWindow () const
SetActiveWindow	HWND SetActiveWindow ()
SetSysModalWindow	HWND SetSysModalWindow ()
SetWindowPos	void SetWindowPos (HWND hWndInsertAfter, const TRect& rect, UINT flags)
SetWindowPos	void SetWindowPos (HWND hWndInsertAfter, int x, int y, int w, int h, UINT flags)

Die gekapselten Funktionen in Tabelle A.7 kontrollieren das Zeichnen in Fenstern, das Gültig- und Ungültigmachen sowie das Aktualisieren von Fenstern.

Tabelle A.7: Fensterzeichenfunktionen

Name	Funktionsdeklaration
FlashWindow	BOOL FlashWindow (BOOL invert)
GetUpdateRect	BOOL GetUpdateRect (TRect& rect, BOOL erase = TRUE) const
Invalidate	void Invalidate (BOOL erase = TRUE)
InvalidateRect	void InvalidateRect (const TRect& rect, BOOL erase = TRUE)
InvalidateRgn	void InvalidateRgn (HRGN hRgn, BOOL erase = TRUE)
LockWindowUpdate	BOOL LockWindowUpdate ()
RedrawWindow	BOOL RedrawWindow (TRect *update, HRGN hUpdateRgn, UINT redrawFlags = RDW_INVALIDATE | RDW_UPDATENOW | RDW_ERASE)
UpdateWindow	void UpdateWindow ()
Validate	void Validate ()
ValidateRect	void ValidateRect (const TRect& rect)
ValidateRgn	void ValidateRgn (HRGN hRgn)

Die Funktionen in Tabelle A.8 kontrollieren das Blättern in Fenstern sowie die Bildlaufleisten.

Tabelle A.8: Funktionen zum Blättern von Fenstern

Name	Funktionsdeklaration
GetScrollPos	int GetScrollPos (int bar)
GetScrollRange	void GetScrollRange (int bar, int &minPos, int &maxPos) const
ScrollWindow	void ScrollWindow (int dx, int dy, const TRect *scroll = 0, const TRect *clip = 0)
ScrollWindowEx	void ScrollWindowEx (int dx, int dy, const TRect *scroll = 0, const TRect *clip = 0, HRGN hUpdateRgn = 0, TRect *update = 0, UINT flags = 0)
SetScrollPos	int SetScrollPos (int bar, int pos, BOOL redraw = TRUE)
SetScrollRange	void SetScrollRange (int bar, int minPos, int maxPos, BOOL redraw = TRUE)
ShowScrollBar	void ShowScrollBar (int bar, BOOL show = TRUE)

Die Funktionen in Tabelle A.9 kontrollieren übergeordnete und untergeordnete Fenster mit Hilfe von Kommando-IDs.

Tabelle A.9: Funktionen mit der ID untergeordneter Fenster

Name	Funktionsdeklaration
CheckDlgButton	void CheckDlgButton (int buttonId, UINT check)
CheckRadioButton	void CheckRadioButton (int firstButtonId, int lastButtonId, int checkButtonId)
GetDlgCtrlID	int GetDlgCtrlID () const
GetDlgItem	HWND GetDlgItem (int childId) const
GetDlgItemInt	UINT GetDlgItemInt (int childId, BOOL* translated, BOOL isSigned) const
GetDlgItemText	int GetDlgItemText (int childId, char far* text, int max) const
GetNextDlgGroupItem	HWND GetNextDlgGroupItem (HWND hWndCtrl, BOOL previous = FALSE) const
GetNextDlgTabItem	HWND GetNextDlgTabItem (HWND HWndCtrl, BOOL previous = FALSE) const
GetParent	HWND GetParent () const
IsChild	BOOL IsChild (HWND) const
IsDlgButtonChecked	UINT IsDlgButtonChecked (int buttonId) const
SetDlgItemInt	void SetDlgItemInt (int childId, UINT value, BOOL isSigned = TRUE) const
SetDlgItemText	void SetDlgItemText (int childId, const char far* text) const

Die Funktionen in Tabelle A.10 kontrollieren Menüs und Menüleistenoperationen.

Tabelle A.10: Menü- und Menüleistenfunktionen

Name	Funktionsdeklaration
DrawMenuBar	void DrawMenuBar ()
GetMenu	HMENU GetMenu ()
GetSystemMenu	HMENU GetSystemMenu (BOOL revert = FALSE)
HiliteMenuItem	BOOL HiliteMenuItem (HMENU, UINT idItem, UINT hilite)
SetMenu	BOOL SetMenu (HMENU)

Die Funktion in Tabelle A.11 kontrolliert Operationen mit der Zwischenablage.

Tabelle A.11: Zwischenablagefunktionen

Name	Funktionsdeklaration
&OpenClipboard	TClipboard &OpenClipboard ()

Die Funktionen in Tabelle A.12 kontrollieren Timer-Operationen.

Tabelle A.12: Timer-Funktionen

Name	Funktionsdeklaration
KillTimer	BOOL KillTimer (UINT timerId)
SetTimer	BOOL SetTimer (UINT timerId, UINT timeout, TIMERPROC proc = 0)

Die Funktionen in Tabelle A.13 kontrollieren Cursor- und Mauszeigeroperationen.

Tabelle A.13: Cursor- und Mauszeigerfunktionen

Name	Funktionsdeklaration
CreateCaret	void CreateCaret (HBITMAP)
CreateCaret	void CreateCaret (int shade, int width, int height)
DestroyCaret	static void DestroyCaret ()
GetCaretBlinkTime	static UINT GetCaretBlinkTime ()
GetCaretPos	static void GetCaretPos (TPoint& pos)
GetCursorPos	static void GetCursorPos (TPoint& pos)
HideCaret	void HideCaret ()
SetCaretBlinkTime	static void SetCaretBlinkTime (WORD milliSecs)
SetCaretPos	static void SetCaretPos (const TPoint& pos)
SetCaretPos	static void SetCaretPos (int x, int y)

Die Funktionen in Tabelle A.14 kontrollieren die Operation von Tastenkürzeln.

Tabelle A.14: Tastenkürzelfunktionen

Name	Funktionsdeklaration
RegisterHotKey	BOOL RegisterHotKey (int idHotKey, UINT modifiers, UINT virtKey)[1]
UnregisterHotKey	BOOL UnregisterHotKey (int idHotKey)[1]
[1] nur WIN32 API	

Die Funktionen in Tabelle A.15 kontrollieren verschiedene Operationen, wie zum Beispiel den Zugriff auf WinHelp.

Tabelle A.15: Hilfs- und Task-Funktionen

Name	Funktionsdeklaration
DragAcceptFiles	void DragAcceptFiles (BOOL accept)
GetWindowTask	HANDLE GetWindowTask () const[1]
GetWindowTask	HTASK GetWindowTask () const[2]
MessageBox	int MessageBox (const char far* text, const char far* caption = 0, UINT type = MB_OK)
WinHelp	BOOL WinHelp (const char far* helpFile, UINT command, DWORD data)
[1] nur WIN32-API [2] nur WIN16-API	

A N H A N G **B**

Strukturen der Windows API

Dieser Anhang beinhaltet mehrere von der Windows API verwendete Strukturen.

Struktur ABC — windows.h

```
typedef struct _ABC {
 int abcA;
 UINT abcB;
 int abcC;
} ABC;
```

Die Struktur ABC enthält die Zeichenbreite in einem TrueType® Font.

Element	Beschreibung
abcA	Zeichenabstand "A". Der Zeichenabstand "A" beschreibt die Distanz, die vor der Darstellung des Zeichenkörpers addiert wird.
abcB	Zeichenabstand "B". Der Zeichenabstand "B" beschreibt die Breite des dargestellten Zeichenkörpers.
abcC	Zeichenabstand "C". Der Zeichenabstand "C" beschreibt die zur gegenwärtigen Position zu addierende Distanz, damit vor dem Zeichenkörper ein Whitespace-Zeichen entsteht.

Die absolute Breite eines Zeichens ergibt sich aus der Summe von *abcA, abcB* und *abcC*. Zum Anzeigen einer Unter- oder Überhanges können entweder *abcA* oder *abcC* negativ sein.

Struktur BITMAP — windows.h

16-Bit-Version:

```
typedef struct tagBITMAP {
 int bmType;
 int bmWidth;
 int bmHeight;
 int bmWidthBytes;
 BYTE bmPlanes;
 BYTE bmBitsPixel;
 void FAR* bmBits;
} BITMAP;
```

32-Bit-Version:

```
typedef struct tagBITMAP {
 LONG bmType;
 LONG bmWidth;
 LONG bmHeight;
 LONG bmWidthBytes;
 WORD bmPlanes;
 WORD bmBitsPixel;
LPVOID bmBits;
} BITMAP;
```

BITMAP definiert die Höhe und Breite, das Farbformat sowie die Bitwerte einer logischen Bitmap.

Element	Beschreibung
bmType	Typ der Bitmap. Bei logischen Bitmaps muß dieses Element Null sein
bmWidth	Breite der Bitmap in Pixeln. Der Wert muß größer als Null sein.
bmHeight	Höhe der Bitmap in Rasterzeilen. Der Wert muß größer als Null sein.
bmWidthBytes	Anzahl der Bytes pro Rasterzeile. Die Graphics Device Interface (GDI) nimmt an, daß die Bitwerte der Bitmap ein Integer-Array (2 Byte) bilden. Deshalb muß der Wert eine gerade Zahl sein.
bmPlanes	Anzahl der Farbebenen der Bitmap.
bmBitsPixel	Anzahl der angrenzenden Farbbits, die in jeder Ebene benötigt werden, um ein Pixel zu definieren.
bmBits	Zeigt auf die Speicherstelle, an der sich die Bitwerte einer Bitmap befinden. Es muß sich dabei um einen Long-Zeiger auf ein Array von Werten mit der Länge eines Bytes handeln.

Gegenwärtig werden monochrome und farbige Bitmap-Formate benutzt. Monochrome Bitmaps benutzen ein Ein-Bit- und Ein-Ebenen-Format. Jede Zeile stellt ein Vielfaches von 16 oder 32 Bits dar. Eine monochrome Bitmap der Höhe *n* ist folgendermaßen organisiert:

```
Scan 0
Scan 1
  ...
```

```
Scan n-2
Scan n-1
```

Monochrome Gerätepixel sind schwarz oder weiß. Wenn ein Bit einer Bitmap 1 ist, wird das zugehörige Pixel umgedreht (weiß). Wenn eine Bitmap 0 ist, wird das zugehörige Pixel umgedreht (schwarz).

Struktur BITMAPCOREHEADER windows.h

```
typedef struct tagBITMAPCOREHEADER {
 DWORD bcSize;
 WORD bcWidth;
 WORD bcHeight;
 WORD bcPlanes;
 WORD bcBitCount;
} BITMAPCOREHEADER;
```

Enthält die Dimension und Farbformat-Information der geräteunabhängigen Bitmap (DIB).

Element	Beschreibung
bcSize	Anzahl der von der Struktur benötigten Bytes.
bcWidth	Breite der Bitmap in Pixeln.
bcHeight	Höhe der Bitmap in Pixeln.
bcPlanes	Anzahl der Ebenen für das Zielgerät. Muß Eins sein.
bcBitCount	Anzahl der Bits pro Pixel. Muß 1, 4, 8 oder 24 sein.

Diese Struktur ist mit einer Farbtafel der Struktur *BITMAPCOREINFO* kombiniert, um eine vollständige Definition der Dimensionen und Farben einer DIB zur Verfügung zu stellen.

Siehe auch Struktur BITMAPCOREINFO, Struktur RGBTRIPLE

Struktur BITMAPCOREINFO

```
typedef struct _BITMAPCOREINFO {
 BITMAPCOREHEADER bmciHeader;
 RGBTRIPLE bmciColors[1];
} BITMAPCOREINFO;
```

Definiert die Dimensionen und Farbinformationen für eine geräteunabhängige Bitmap (DIB).

Element	Beschreibung
bmciHeader	Struktur mit Dimensionen und Farbinformationen für die DIB.
bmciColors	Struktur, die die Farben einer Bitmap definiert.

Eine DIB enthält zwei Teile einer Struktur vom Typ *BITMAPCOREINFO*. Diese definieren die Dimensionen und Farben der Bitmap, sowie ein Byte-Array, das die Pixel der Bitmap beschreibt. Die Bits des Arrays sind hintereinandergereiht, aber jede Zeile muß mit Nullen gefüllt werden, um an LONG zu grenzen. Der Ursprung der Bitmap befindet sich in der unteren linken Ecke.

Das Element *bcBitCount* Element der Struktur *BITMAPCOREHEADER* bestimmt die Anzahl der Bits, die jedes Pixel und die maximale Anzahl von Farben in der Bitmap definieren. Diesem Element kann einer der folgenden Werte zugeordnet sein:

Wert	Beschreibung
1	Die Bitmap ist monochrom und die Elemente von *bmciColors* enthalten zwei Einträge. Jedes Bit im Bitmap-Array stellt ein Pixel dar. Wenn das Bit nicht gesetzt ist, wird das Pixel mit der Farbe des ersten Tabelleneintrages in *bmciColors* angezeigt. Wenn das Bit gesetzt ist, hat das Pixel die Farbe des zweiten Tabelleneintrages.
4	Die Bitmap enthält maximal 16 Farben und das Element *bmciColors* enthält bis zu 16 Einträge. Jedes Pixel der Bitmap wird mittels eines 4-Bit-Indices in der Farbtabelle dargestellt. Wenn z.B. das erste Byte der Bitmap 0x1F ist, stellt das Byte zwei Pixel dar. Das erste Pixel besitzt die Farbe des zweiten Tabelleneintrags und das zweite Pixel die Farbe des sechzehnten Tabelleneintrags.
8	Die Bitmap stellt maximal 256 Farben dar und das Element *bmciColors* enthält bis zu 256 Einträge. In diesem Fall wird jedes Byte durch ein einziges Pixel dargestellt.
24	Die Bitmap enthält maximal 2 Farben und das Element *bmciColors* ist NULL. Jedes 3-Byte-Tripel im Bitmap-Array stellt die relative Intensität eines Pixels in Rot, Grün beziehungsweise Blau dar.

Die Farben in der Tabelle *bmciColors* sollten in der Reihenfolge ihrer Wichtigkeit erscheinen. Alternativ dazu können für Funktionen, die DIBs benutzen, die Elemente *bmciColors* Unsigned Integers der Länge 16 Bit enthalten. Diese bezeichnen Indices in der gegenwärtig realisierten logischen Palette anstatt von expliziten RGB-Werten. In diesem Fall muß eine die Bitmap benutzende Applikation die DIB-Funktionen *CreateDIBitmap*, *CreateDIBPatternBrush* bzw. *CreateDIBSection* mit dem in DIB_PAL_COLORS gesetzten Parameter *iUsage* verwenden.

Hinweis

Das Element *bmciColors* sollte keine Indices der Palette verwenden, wenn die Bitmap in eine Datei gespeichert wurde oder in eine andere Applikation übertragen wurde. Die Farbtafel der Bitmap sollte so lange explizit

Werte der RGB verwenden, bis die Applikation die Bitmap exclusiv benutzt und kontrolliert.

Struktur BITMAPINFO windows.h

16-Bit-Version:

```
typedef struct tagBITMAPINFO {
 BITMAPINFOHEADER bmiHeader;
 RGBQUAD bmiColors[1];
} BITMAPINFO;
```

Definiert die Dimensionen und Farbinformationen für eine geräteunabhängige Bitmap (DIB).

Element	Beschreibung
bmiHeader	Struktur mit Dimensionen und Farbinformationen für eine DIB.
bmciColors	Struktur, die die Farben einer Bitmap definiert.

Eine DIB enthält eine Struktur vom Typ *BITMAPINFO*. Diese definiert die Dimensionen und Farben der Bitmap, sowie eine Byte-Array, das die Pixel der Bitmap beschreibt. Die Bits des Arrays sind hintereinandergereiht, aber jede Zeile muß mit Nullen gefüllt werden, um an LONG zu grenzen. Die Segmentgrenzen können sich an einer beliebigen Stelle der Bitmap befinden. Der Ursprung der Bitmap befindet sich in der unteren linken Ecke.

Das Element *biBitCount* der Struktur *BITMAPINFOHEADER* bestimmt die Anzahl der Bits, die jedes Pixel und die maximale Anzahl von Farben in der Bitmap definieren. Diesem Element kann einer der folgenden Werte zugeordnet sein:

Wert	Beschreibung
1	Die Bitmap ist monochrom und die Elemente von *bmciColors* enhalten zwei Einträge. Jedes Bit im Bitmap-Array stellt ein Pixel dar. Wenn das Bit nicht gesetzt ist, wird das Pixel mit der Farbe des ersten Tabelleneintrages in *bmciColors* angezeigt; wenn das Bit gesetzt ist, hat das Pixel die Farbe des zweiten Tabelleneintrages.
4	Die Bitmap enthält maximal 16 Farben und das Element *bmciColors* enthält 16 Einträge. Jedes Pixel der Bitmap wird mittels eines 4-Bit-Indices in der Farbtabelle dargestellt. Wenn z.B. das erste Byte der Bitmap 0x1F ist, stellt das Byte zwei Pixel dar. Das erste Pixel besitzt die Farbe des zweiten Tabelleneintrags und das zweite Pixel die Farbe des sechzehnten Tabelleneintrags.
8	Die Bitmap stellt maximal 256 Farben dar und das Element *bmciColors* enthält 256 Einträge. In diesem Fall wird jedes Byte durch ein einziges Pixel dargestellt.
16	Die Bitmap besitzt ein Maximum von 2 hoch 16 Farben. Das Element *biCompression* von MITMAPINFOHEADER muß BI_FITFIELDS sein. Das Element *bmiColours* enthält 3 DWORD Farbmasken, welche die Rote, Grüne bzw. Blaue Komponenete jedes Pixels darstellen. Die in der Maske DWORD gesetzten Bits müssen fortlaufend sein und sollten die Bits einer anderen Maske

	nicht überlappen. Es müssen nicht alle Bits eines Pixels verwendet werden. Jedes WORD eines Arrays stellt ein einziges Pixel dar.
24	Die Bitmap enthält maximal 2 Farben und das Element *bmciColors* ist NULL. Jedes 3-Byte-Tripel im Bitmap-Array stellt die relative Intensität eines Pixels in Rot, Grün beziehungsweise Blau dar.
32	Die Bitmap enthält maximal 2 Farben. Das Element *biCompression* von MITMAPINFOHEADER muß BI_FITFIELDS sein. Das Element *bmiColours* enthält 3 DWORD Farbmasken, welche die Rote, Grüne bzw. Blaue Komponenete jedes Pixels darstellen. Die in der Maske DWORD gesetzten Bits müssen fortlaufend sein und sollten die Bits einer anderen Maske nicht überlappen. Es müssen nicht alle Bits eines Pixels verwendet werden. Jedes WORD eines Arrays stellt ein einziges Pixel dar.

Das Element *biClrUsed* der Struktur BITMAPINFOHEADER spezifiziert die Anzahl der Farbindices in der gegenwärtig von der Bitmap benutzten Farbtabelle dar. Wenn das Element *biClrUsed* auf Null gesetzt ist, verwendet die Bitmap die zum Wert des Elements *biBitCount* gehörende maximale Anzahl von Farben dar. Die Farben in der Tabelle *bmciColors* sollten in der Reihenfolge ihrer Wichtigkeit erscheinen.

Alternativ dazu können für Funktionen, die DIBs benutzen, die Elemente *bmiColors* unsigned integers der Länge 16 Bit enthalten. Diese bezeichnen Indices in der gegenwärtig realisierten logischen Palette anstatt von expliziten RGB-Werten. In diesem Fall muß eine die Bitmap benutzende Applikation die DIB-Funktionen *CreateDIBitmap*, *CreateDIBPatternBrush* bzw. *CreateDIBSection* mit dem in DIB_PAL_COLORS gesetzten Parameter *wUsage* verwenden.

Hinweis

Das Element *bmciColors* sollte keine Indices der Palette verwenden, wenn die Bitmap in eine Datei gespeichert wurde oder in eine andere Applikation übertragen wurde. Die Farbtafel der Bitmap solle so lange explizit Werte der RGB verwenden, bis die Applikation die Bitmap exclusiv benutzt und kontrolliert.

Struktur BITMAPINFOHEADER windows.h

```
typedef struct tagBITMAPINFOHEADER {
 DWORD biSize;
 LONG biWidth;
 LONG biHeight;
 WORD biPlanes;
 WORD biBitCount;
 DWORD biCompression;
 DWORD biSizeImage;
 LONG biXPelsPerMeter;
 LONG biYPelsPerMeter;
 DWORD biClrUsed;
```

```
 DWORD biClrImportant;
} BITMAPINFOHEADER;
```

Enthält die Dimensionen und Farbinformationen für eine geräteunabhängige Bitmap (DIB).

Element	Beschreibung
biSize	Anzahl der von dieser Struktur benötigten Bytes
biWidth	Breite der Bitmap in Pixeln.
biHeight	Höhe der Bitmap in Pixeln. Wenn *biHeight* negativ ist, befindet sich der Ursprung der Bitmap in der unteren linken Ecke und die Höhe ist der Absolutwert von *biHeight*.
biPlanes	Legt die Anzahl der Ebenen für das Zielgerät fest. Dieses Element muß auf 1 gesetzt sein.
biBitCount	Legt die Anzahl der Bits pro Pixel fest. Dieser Wert muß 1, 4, 8, 16 (Nur 16-Bit), 24 oder 32 (nur 32-Bit) sein.
biCompression	Bezeichnet den Kompressionstyp für eine komprimierte Bitmap. Es können folgende Werte auftreten: **Wert** — **Bedeutung** BI_RGB — Unkomprimiert. BI_RLE8 — Legt das Format Run-Length Encoded für Bitmaps mit 8 Bits pro Pixel fest. BI_RLE4 — Legt das Format Run-Length Encoded für Bitmaps mit 4 Bits pro Pixel fest. BI_BITFIELDS — Spezifiziert die Bitmap als nicht komprimiert und die Farbtabelle mit 3 DWORD Farbmasken, die für jedes Pixel die roten, grünen bzw. blauen Komponenten enthalten. Das ist gültig, wenn mit Bitmaps 16 und 32 Bits pro Pixel gearbeitet wird.
biSizeImage	Größe des Bildes in Bytes. Kann 0 sein, wenn die Bitmap im Format BI_RGB ist.
biXPelsPerMeter	Legt die horizontale Auflösung des Zielgerätes für die Bitmap in Pixel pro Meter fest. Verwendet diesen Wert, um von der Ressourcen-Gruppe diejenige Bitmap auszuwählen, das am besten zu den Eigenschaften des jeweiligen Gerätes paßt.
biYPelsPerMeter	Legt die vertikale Auflösung des Zielgerätes in Pixel pro Meter fest.
biClrUsed	Größe des Bildes in Bytes. Kann 0 sein, wenn die Bitmap im Format BI_RGB ist. Legt die horizontale Auflösung des Zielgerätes für die Bitmap in Pixel pro Meter fest. Verwendet diesen Wert, um von der Ressourcen-Gruppe diejenige Bitmap auszuwählen, das am besten zu den Eigenschaften des jeweiligen Gerätes paßt. Legt die vertikale Auflösung des Zielgerätes in Pixel pro Meter fest. Legt die Anzahl der Farbindices in der Farbtabelle fest, die gegenwärtig von der Bitmap benutzt werden. Wenn der Wert Null ist, verwendet die Bitmap die zum Wert des Elementes *biBitCounter* gehörende maximale Anzahl von Farben. Wenn bei 16-Bit-Applikationen *biClrUsed* nicht Null ist, wird die aktuelle Anzahl von Farben bezeichnet, auf die die Grafikmaschine oder der Gerätetreiber zugreifen, falls das Element *biBitCount* kleiner als 24 ist. Wenn *biBitCount* auf 24 gesetzt ist, spezifiziert *biClrUsed* die Größe der referenzierten Farbtabelle, die zur Optimierung der Leistungsfähigkeit von Farbpaletten benutzt wird. Wenn bei 32-Bit-Applikationen *biClrUsed* nicht Null und das Element *biBitCount* kleiner als sechzehn sind, bezeichnet das Element *biClrUsed* die aktuelle Anzahl von Farben der Grafikmaschine oder des Gerätetreibers. Wenn *biBitCount* größer oder gleich sechzehn ist, dann spezifiziert das Element *biClrUsed* die Größe der Farbtabelle, die zur Optimierung der Leistungsfähigkeit von Farbpaletten unter Windows benutzt wird. Wenn *biBitCount* gleich 16 oder 32 ist, startet die optimale Farbpalette sofort nach der Maske 3 DWORD. Wenn die Bitmap gepackt ist (d.h., wenn in einer Bitmap das Bitmap-Array sofort auf den Header

biClrImportant	*BITMAPINFO* folgt und wenn die Bitmap durch einen einzigen Zeiger referenziert wird), muß des Element *biClrUsed* auf Null oder auf die aktuelle Größe der Farbtabelle gesetzt werden. Bezeichnet die Anzahl der Farbindices, die für die Anzeige der Grafik als wichtig angesehen werden. Wenn der Wert Null ist, sind alle Farben wichtig.

typedef COLORREF windows.h

typedef DWORD COLORREF;

Ein 32-Bit-Wert, der zur Spezifizierung einer Farbe vom Typ RGB benutzt wird. Der Wert *COLORREF* ist im folgenden hexadezimalen Format:

```
0x00bbggrr
```

Das niederwertige byte (*rr*) enthält einen Wert für die relative Intensität von Rot; das zweite Byte (*gg*) enthält einen Wert von Grün; das dritte Byte (*bb*) enthält einen Wert für Blau. Das vierte Byte muß Null sein. Das Makro RGB kann folgendermaßen zum Setzen dieser Werte verwendet werden:

RGB(red, green, blue)

Jeder Farbparameter kann zwischen 0x0 und 0xFF liegen.

Struktur COMPAREITEMSTRUCT windows.h

```
typedef struct tagCOMPAREITEMSTRUCT {
 UINT  CtlType;
 UINT  CtlID;
 HWND  hwndItem;
 UINT  itemID1;
 DWORD itemData1;
 UINT  itemID2;
 DWORD itemData2;
} COMPAREITEMSTRUCT;
```

Liefert die Bezeichner und die von der Applikation unterstützte Daten in einer sortierten vom Benutzer gezeichneten Auswahlliste oder einem Kombinationsfenster. Die folgende Tabelle beschreibt die einzelnen Elemente.

Element	Beschreibung
CtlType	Entweder ODT_LISTBOX für eine vom Benutzer definierte Auswahlliste oder ODT_COMBOBOX für ein vom Benutzer definiertes Kombinationsfenster.
CtlID	Bezeichner für Auswahlliste oder Kombinationsfenster.

hwndItem	Dialogelementbezeichner.
itemID1	Index des ersten Eintrags in der Auswahlliste oder dem Kombinationsfenster, mit dem verglichen wird.
itemData1	Applikationsunterstützte Daten des ersten Eintrags, mit dem verglichen wird.
itemID2	Index des zweiten Eintrags in der Auswahlliste oder dem Kombinationsfenster, mit dem verglichen wird.
itemData2	Applikationsunterstützte Daten für den zweiten Eintrag, mit dem verglichen wird.

Struktur DELETEITEMSTRUCT windows.h

```
typedef struct tagDELETEITEMSTRUCT {
 UINT CtlType;
 UINT CtlID;
 UINT itemID;
 HWND hwndItem;
 UINT itemData;
} DELETEITEMSTRUCT;
```

Beschreibt einen gelöschten vom Benutzer gezeichneten Eintrag in einer Auswahlliste oder einem Kombinationsfenster. Die folgende Tabelle beschreibt die Elemente.

Element	Beschreibung
CtlType	Entweder ODT_LISTBOX für eine vom Benutzer gezeichnete Auswahlliste oder ODT_COMBOBOX für ein vom Benutzer gezeichnetes Kombinationsfenster.
CtlID	Bezeichner der Auswahlliste oder des Kombinationsfensters.
itemID	Indes für den gelöschten Eintrag in der Auswahlliste bzw. dem Kombinatiosfenster.
hwndItem	Bezeichner für ein Dialogelement.
itemData	Benutzerdefinierter Dateneintrag.

Struktur DEVMODE windows.h

16-Bit-Version:

```
typedef struct tagDEVMODE {
 char  dmDeviceName[CCHDEVICENAME];
 UINT  dmSpecVersion;
 UINT  dmDriverVersion;
 UINT  dmSize;
 UINT  dmDriverExtra;
 DWORD dmFields;
 int   dmOrientation;
 int   dmPaperSize;
 int   dmPaperLength;
```

```
 int   dmPaperWidth;
 int   dmScale;
 int   dmCopies;
 int   dmDefaultSource;
 int   dmPrintQuality;
 int   dmColor;
 int   dmDuplex;
 int   dmYResolution;
 int   dmTTOption;
} DEVMODE;
```

32-Bit-Version:

```
typedef struct _devicemode {
 TCHAR  dmDeviceName[32];
 WORD   dmSpecVersion;
 WORD   dmDriverVersion;
 WORD   dmSize;
 WORD   dmDriverExtra;
 DWORD  dmFields;
 short  dmOrientation;
 short  dmPaperSize;
 short  dmPaperLength;
 short  dmPaperWidth;
 short  dmScale;
 short  dmCopies;
 short  dmDefaultSource;
 short  dmPrintQuality;
 short  dmColor;
 short  dmDuplex;
 short  dmYResolution;
 short  dmTTOption;
 short  dmCollate;
 TCHAR  dmFormName[32];
 WORD   dmUnusedPadding;
 USHORT dmBitsPerPel;
 DWORD  dmPelsWidth;
 DWORD  dmPelsHeight;
 DWORD  dmDisplayFlags;
 DWORD  dmDisplayFrequency;
} DEVMODE;
```

Enthält Daten über die Initialisierung und die Umgebung von Druckertreibern. Die folgende Tabelle beschreibt jedes Element.

Element	Beschreibung
dmDeviceName	Name des Gerätes, z.B. für die Treiberunterstützung.
dmSpecVersion	Versionsnummer dieser Struktur.
dmDriverVersion	Versionsnummer des Druckertreibers.
dmSize	Größe dieser Struktur in Bytes, ausgenommen *dmDriverData*. Wird benutzt um die Länge einer

Struktur herauszufinden, ohne sich mit verschiedenen Versionen beschäftigen zu müssen.

dmDriverExtra
Größe des optionalen Elementes *dmDriverData* in Bytes. Wenn eine Applikation keine gerätespezifischen Informationen verwendet, sollte sie dieses Element auf Null setzen.

dmFields
Enthält Flags, die anzeigen, welche der verbleibenden Elemente in dieser Struktur bereits initialisiert wurden. Jede Kombination der folgenden Werte ist möglich

DM_ORIENTATION	DM_DEFAULTSOURCE
DM_PAPERSIZE	DM_PRINTQUALITY
DM_PAPERLENGTH	DM_COLOR
DM_PAPERWIDTH	DM_DUPLEX
DM_SCALE	DM_YRESOLUTION
DM_COPIES	DM_TTOPTION

dmOrientation
Papierausrichtung. Kann DMORIENT_PORTRAIT oder DMORIENT_LANDSCAPE sein.

dmPaperSize
Papiergröße. Kann auf Null gesetzt sein, wenn die Papierlänge und -breite in *dmPaperLength* und *dmPaperWidth* festgelegt werden. Andernfalls einer der folgenden Werte:

Wert	**Bedeutung**
DMPAPER_FIRST	DMPAPER_LETTER
DMPAPER_LETTER	Letter, 8 1/2 x 11 inch.
DMPAPER_LETTERSMALL	Letter Small, 8 1/2 x 11 inch.
DMPAPER_TABLOID	Tabloid, 11 x 17 inch.
DMPAPER_LEDGER	Ledger, 17 x 11 inch.
DMPAPER_LEGAL	Legal, 8 1/2 x 14 inch.
DMPAPER_STATEMENT	Statement, 5 1/2 x 8 1/2 inch.
DMPAPER_EXECUTIVE	Executive, 7 1/2 x 10 1/2 inch.
DMPAPER_A3	A3, 297 x 420 mm
DMPAPER_A4	A4, 210 x 297 mm
DMPAPER_A4SMALL	A4 Small, 210 x 297 mm
DMPAPER_A5	A5, 148 x 210 mm
DMPAPER_B4	B4, 250 x 354 mm
DMPAPER_B5	B5, 182 x 257 mm
DMPAPER_FOLIO	Folio, 8 1/2 x 13 inch.
DMPAPER_QUARTO	Quarto, 215 x 275 mm
DMPAPER_10X14	10 x 14 inch.
DMPAPER_11X17	11 x 17 inch.
DMPAPER_NOTE	Note, 8 1/2 x 11 inch.
DMPAPER_ENV_9	Envelope #9, 3 7/8 x 8 7/8 inch.
DMPAPER_ENV_10	Envelope #10, 4 1/8 x 9 1/2 inch.
DMPAPER_ENV_11	Envelope #11, 4 1/2 x 10 3/8 inch.
DMPAPER_ENV_12	Envelope #12, 4 1/2 x 11 inch.
DMPAPER_ENV_14	Envelope #14, 5 x 11 1/2 inch.
DMPAPER_CSHEET	C size sheet
DMPAPER_DSHEET	D size sheet
DMPAPER_ESHEET	E size sheet
DMPAPER_ENV_DL	Envelope DL, 110 x 220 mm
DMPAPER_ENV_C3	Envelope C3, 324 x 458 mm
DMPAPER_ENV_C4	Envelope C4, 229 x 324 mm
DMPAPER_ENV_C5	Envelope C5, 162 x 229 mm
DMPAPER_ENV_C6	Envelope C6, 114 x 162 mm
DMPAPER_ENV_C65	Envelope C65, 114 x 229 mm
DMPAPER_ENV_B4	Envelope B4, 250 x 353 mm
DMPAPER_ENV_B5	Envelope B5, 176 x 250 mm

	DMPAPER_ENV_B6	Envelope B6, 176 x 125 mm
	DMPAPER_ENV_ITALY	Envelope, 110 x 230 mm
	DMPAPER_ENV_MONARCH	Envelope Monarch, 3 7/8 x 7 1/2 inch.
	DMPAPER_ENV_PERSONAL	Envelope, 3 5/8 x 6 1/2 inch.
	DMPAPER_FANFOLD_US	U.S. Standard Fanfold, 14 7/8 x 11 inch.
	DMPAPER_FANFOLD_STD_GERMAN	German Standard Fanfold, 8 1/2 x 12 inch.
	DMPAPER_FANFOLD_LGL_GERMAN	German Legal Fanfold, 8 1/2 x 13 inch.
	DMPAPER_LAST	German Legal Fanfold, 8 1/2 x 13 inch.
	DMPAPER_USER	Benutzerdefiniert

dmPaperLength
Papierlänge in Zehntelmillimetern. Überschreibt *dmPaperSize*.

dmPaperWidth
Papierbreite in Zehntelmillimetern. Überschreibt *dmPaperSize*.

dmScale
Skalierfaktor für die Druckausgabe. Aktuelle Seitengröße wird von der physikalischen Seitengröße berechnet mit dmScale/100.

dmCopies
Legt die Anzahl der gedruckten Kopien fest, wenn das Gerät Mehrfachkopien von Seiten unterstützt.

dmDefaultSource
Automatischer Schacht für den Papiereinzug. Dieses Element kann einen der folgenden Werte haben:

DMBIN_AUTO	DMBIN_LOWER
DMBIN_CASSETTE	DMBIN_MANUAL
DMBIN_ENVELOPE	DMBIN_MIDDLE
DMBIN_ENVMANUAL	DMBIN_ONLYONE
DMBIN_FIRST	DMBIN_SMALLFMT
DMBIN_LARGECAPACITY	DMBIN_TRACTOR
DMBIN_LARGEFMT	DMBIN_UPPER
DMBIN_LAST	

mPrintQuality
Druckerauflösung. Kann einer der folgenden Werte sein:
DMRES_HIGH
DMRES_MEDIUM
DMRES_LOW
DMRES_DRAFT
Wenn ein anderer (positiver) Wert gegeben ist, bezeichnet er Dots Per Inch (DPI) und ist geräteunabhängig. Wenn der Drucker *dmYResolution* initialisiert, dann spezifiziert *dmPrintQuality* die X-Auflösung des Druckers in DPI.

dmColor
Farbige oder monochrome Ausgabe. Kann DMCOLOR_COLOR oder DMCOLOR_MONOCHROME sein.

dmDuplex
Duplex (doppelseitiger) Ausdruck für Drucker, die das unterstützen. Dieses Element kann einen der folgenden Werte haben:
DMDUP_SIMPLEX
DMDUP_HORIZONTAL
DMDUP_VERTICAL

dmYResolution
Y-Auflösung für den Drucker in DPI. Wenn der Drucker *dmYResolution* initialisiert, spezifiziert *dmPrintQuality* die X-Auflösung des Druckers in DPI.

dmTTOption
Ausdrucken von TrueType-Schriften. Kann einen der folgenden Werte haben:

DMTT_BITMAP	Druckt TrueType-Schriften als Grafiken. Die Standardeinstellung für Matrixdru
DMTT_DOWNLOAD	Senden von TrueType-Fonts als soft Fonts an den Drucker. Voreinstellung für Drucker von Hewlett-Packard,die die Printer Control Language (PCL) verwenden.
DMTT_SUBDEV	Ersetzen von TrueType-Fonts durch Gerätefonts. Voreinstellung bei PostScript-Druckern.

dmUnusedPadding
(nur 32-Bit) Wird benutzt, um die Struktur am Ende von DWORD-Grenze auszurichten.

dmCollate
Vergleichen beim Drucken mehrerer Kopien. Kann einer der folgenden Werte sein:

DMCOLLATE_TRUE	Sortieren beim Drucken mehrerer Kopien.
DMCOLLATE_FALSE	NICHT sortieren beim Drucken mehrerer Kopien.

dmFormName	(nur 32-Bit) Bestimmt den Namen des benutzten Formats, z.B. »Letter« oder »Letter«.
dmBitsPerPel	(nur 32-Bit) Farbauflösung für das Anzeigegerät, in Bits pro Pixel.
dmPelsWidth	(nur 32-Bit) Breite der sichtbaren Geräteoberfläche in Pixeln.
dmPelsHeight	(nur 32-Bit) Höhe der sichtbaren Geräteoberfläche in Pixeln.
dmDisplayFlags	(nur 32-Bit) Anzeigemodus des Gerätes. Kann einer der folgenden Werte sein:
	Wert — **Bedeutung**
	DM_GRAYSCALE — Anzeige ohne Farbe. Wenn dieses Flag nicht gesetzt ist, wird Farbmodus angenommen.
	DM_INTERLACED — Anzeigemodus interlaced. Wenn dieses Flag nicht gesetzt ist, wird non-interlaced angenommen.
dmDisplayFrequency	(nur 32-Bit) Bestimmt die Frequenz des Anzeigegerätes in Hertz.

Struktur DRAWITEMSTRUCT

windows.h

```
typedef struct tagDRAWITEMSTRUCT {
 UINT CtlType;
 UINT CtlID;
 UINT itemID;
 UINT itemAction;
 UINT itemState;
 HWND hwndItem;
 HDC hDC;
 RECT rcItem;
 DWORD itemData;
} DRAWITEMSTRUCT;
```

Stellt dem Besitzerfenster Informationen zur Verfügung, die es braucht, um zu entscheiden, wie ein vom Besitzer gezeichnetes Dialogelement oder eine Menüoption gezeichnet werden sollen. Das Besitzerfenster des vom Besitzer gezeichneten Dialogelements oder Menüoption erhält als Zeiger auf diese Struktur den Parameter *lParam* der Meldung `WM_DRAWITEM`.

Element	Beschreibung
CtlType	Bestimmt den Elementtyp. Dieses Element kann einen der folgenden Werte annehmen:
	ODT_BUTTON — Vom Besitzer gezeichneter Schalter
	ODT_COMBOBOX — Vom Besitzer gezeichnetes Kombinationsfenster
	ODT_LISTBOX — Vom Besitzer gezeichnetes Listenfenster
	ODT_MENU — Vom Besitzer gezeichnete Menüoption
CtlID	Bestimmt den Bezeichner des Kombinationsfensters, des Listenfensters oder der Schaltfläche. Dieses Element kann nicht für eine Menüoption benutzt werden.
itemID	Bestimmt den Bezeichner einer Menüoption für eine Menüoption oder den Index des Eintrags in einem Listenfeld oder einem Kombinationsfenster. Für ein leeres Listenfeld oder Kombinationsfenster kann dieses Element auch - 1 sein. Dies erlaubt der Anwendung, nur das Fokus-Rechteck zu zeichnen, das vom Element *rcItem* bestimmt wird, obwohl das Dialogelement leer ist. Dies zeigt dem Benutzer an, ob das Listenfeld oder das Kombinationsfenster den Focus besitzt.

Ob das Rechteck so gezeichnet wird, als ob sich der Fokus im Listenfeld befände oder im Kombinationsfenster, wird dadurch bestimmt, wie die Bits im Element *itemAction* gesetzt sind.

itemAction
Bestimmt, was gezeichnet werden muß. Das Element kann einen oder mehrere der folgenden Werte haben:

- ODA_DRAWENTIRE — Das gesammte Dialogelement muß gezeichnet werden.
- ODA_FOCUS — Das Dialogelement hat den Tastaturfokus erhalten oder verloren. Das Element *itemState* sollte überprüft werden, um festzustellen, ob das Dialogelement den Fokus hat.
- ODA_SELECT — Der Auswahlstatus hat sich geändert. Das Element *itemState* sollte überprüft werden, um den neuen Auswahlstatus zu überprüfen.

itemState
Bestimmt den optischen Zustand des Elements nach dem Zeichnen. Dieses Element kann eine Kombination folgender Werte sein:

- ODS_CHECKED — Die Menüoption muß geprüft werden. Dieses Bit wird nur in Menüs benutzt.
- ODS_DISABLED — Die Menüoption wird gesperrt gezeichnet.
- ODS_FOCUS — Auf dieser Menüoption befindet sich der Tastaturfokus.
- ODS_GRAYED — Diese Menüoption wird Grau gezeichnet. Dieses Bit wird nur in Menüs benutzt.
- ODS_SELECTED — Der Status dieses Menüelements ist ausgewählt.

hwndItem
Bezeichnet das Dialogelement für Kombinationsfenster, Listenfelder und Schalter. Für Menüs identifiziert dieses Element das Menü, das das Element enthält.

hDC
Bezeichnet einen Gerätekontext (DC); dieser DC muß bei Zeichenoperationen am Dialogfeld benutzt werden.

rcItem
Definiert ein Rechteck, das die Grenzen des zu zeichnenden Dialogelements definiert. Dieses Rechteck ist in dem DC enthalten, der durch das *hDC* Element definiert ist. Alles, außer Menüeoptionen, was das Besitzerfenster in den DC für Kombinationsfenster, Listenfelder und Schaltflächen zeichnet, wird automatisch abgeschnitten. Beim Zeichnen von Menüelementen darf das Besitzerfenster nicht außerhalb des Rechtecks zeichnen, das durch das Element *rcItem* definiert wurde.

itemData
Bestimmt den durch die Anwendung definierten 32-bit Wert, der mit der Menüoption verbunden ist. Zur Kontrolle bestimmt dieser Parameter den Wert, der dem Listenfeld oder dem Kombinationsfenster von der Meldung LB_SETITEMDATA oder CB_SETITEMDATA zuletzt zugewiesen wurde. Wenn das Listenfeld oder das Kombinationsfenster die Eigenschaften LBS_HASSTRINGS oder CBS_HASSTRINGS hat, ist dieser Anfangswert 0. Andernfalls ist dies derjenige Wert, der dem Listenfeld oder dem Kombinationsfenster vom Parameter *lParam* übergeben wurde.

Struktur FINDREPLACE

windows.h

```
typedef struct {
 DWORD        lStructSize;
 HWND         hwndOwner;
 HINSTANCE    hInstance;
 DWORD        Flags;
 LPSTR        lpstrFindWhat;
 LPSTR        lpstrReplaceWith;
 WORD         wFindWhatLen;
 WORD         wReplaceWithLen;
```

```
  DWORD        lCustData;
  LPFRHOOKPROC lpfnHook;
  LPCSTR       lpTemplateName;
} FINDREPLACE;
```

Enthält die Initialisierungsinformationen zum Initialisieren der systemdefinierten Dialogfelder für Suchen und Ersetzen. Die folgende Tabelle beschreibt die Elemente:

Element	Beschreibung
lStructSize	Länge der Struktur in Bytes.
hwndOwner	Besitzerfenster des Dialogfeldes. Kann jedes beliebige Fenster-Handle sein, nicht jedoch NULL.
hInstance	Datenblock, der eine Dialogfeldvorlage, die von *lpstrTemplateName* bestimmt wird. Wird nur benutzt, wenn das Element *Flags* auf FR_ENABLETEMPLATE gesetzt ist; andernfalls wird es ignoriert.
Flags	Flag zum Initialisieren von Dialogfeldern. Kann aus folgenden Werten kombiniert sein: **Wert** — **Bedeutung** FR_DIALOGTERM — Zeigt an, daß das Dialogfeld sich schließt. FR_DOWN — Wenn dieses Flag gesetzt ist, wird nach unten hin gesucht. Ist das Flag nicht gesetzt, wird nach oben gesucht. FR_ENABLEHOOK — Ermöglicht die Hook-Funktion, die im Element *lpfnHook* der Struktur festgelegt wird. Wird nur zum Initialisieren des Dialogfensters benutzt. FR_ENABLETEMPLATE — Erstellt das Dialogfeld und benutzt dabei die Vorlage, das duch die Elemente *hInstance* und *lpTemplateName* identifiziert wird. Dieses Flag wird nur zum Initialisieren des Dialogfeldes benutzt. FR_ENABLETEMPLATEHANDLE — Zeigt an, daß das Element *hInstance* einen Datenblock identifiziert, der eine zuvor geladene Dialogfeldvorlage enthält. Das Element *lpTemplateName* wird ignoriert, wenn dieses Flag gesetzt ist. FR_FINDNEXT — Die Anwendung soll nach dem nächsten Vorkommen der Zeichenkette suchen, die vom Element *lpstrFindWhat* gesetzt wird. FR_HIDEUPDOWN — Verbirgt das Markierungsfeld für die Richtung und für die Steuerelemente Auf und Ab. FR_HIDEMATCHCASE — Verbirgt das Markierungsfeld Groß-/ Kleinschreibung beachten. FR_HIDEWHOLEWORD — Verbirgt das Markierungsfeld Als Wort. FR_MATCHCASE — Bei der Suche werden Groß- und Kleinschreibung beachtet. FR_NOMATCHCASE — Sperrt das Markierungsfeld Groß-/ Kleinschreibung beachten FR_NOUPDOWN — Sperrt die Richtungsschaltfelder. FR_NOWHOLEWORD — Sperrt das Markierungsfeld als Wort. FR_REPLACE — Ersetzt das aktuelle Vorkommen der von *lpstrFindWhat* definierten Zeichenkette durch die durch *lpstrReplaceWith* definierte Zeichenkette. FR_REPLACEALL — Ersetzt alle Vorkommen der von *lpstrFindWhat* definierten Zeichenkette durch die durch *lpstrReplaceWith* definierte Zeichenkette. FR_SHOWHELP — Zeigt die Schaltfläche Hilfe. Das Element *hwndOwner* darf nicht NULL sein, wenn diese Option gesetzt ist. FR_WHOLEWORD — Überprüft das Markierungsfeld Als Wort.
lpstrFindWhat	Die zu suchende Zeichenkette. Wenn beim Aufruf des Dialogfelds eine Zeichenkette festgelegt ist, initialisiert das Dialogfeld das Textfeld Suchen nach mit dieser Zeichenkette. Wenn das Flag FR_FINDNEXT beim Öffnen des Dialogfelds gesetzt ist, sucht die Anwendung nach einem

	Vorkommen dieser Zeichenkette. Dabei benutzt es die Flags FR_DOWN, FR_WHOLEWORD und FR_MATCHCASE, um die Art der Suche und die Suchrichtung zu bestimmen. Die Anwendung muß der Zeichenkette einen Puffer zuweisen, der mindestens 80 Zeichen lang sein muß.
lpstrReplaceWith	Die Zeichenkette, mit der der Suchtext ersetzt wird.
wFindWhatLen	Länge des Puffers, auf den *lpstrFindWhat* zeigt; in Bytes.
wReplaceWithLen	Länge des Puffers, auf den *lpstrFindWith* zeigt; in Bytes.
lCustData	Anwendungsdefinierte Daten, die an die Hook-Funktion *lpfnHook* übergeben werden.
lpfnHook	Zeigt auf eine Hook-Funktion, die Meldungen für das Dialogfeld verarbeitet. Eine Anwendung muß das Flag FR_ENABLEHOOK in den *Flag* Elementen bestimmen, ansonsten ignoriert das System dieses Strukturelement. Gibt den Wert FALSE zurück, um eine Meldung an die Standard-Dialogprozedur zu übergeben, oder TRUE um die Meldung zu verwerfen.
lpstrTemplateName	Zeigt auf einen nullterminierten String, der die Dialogfeldvorlagen-Ressource benennt, die die Standard-Dialogfeldvorlage ersetzen soll.

Struktur GLYPHMETRICS

windows.h

```
typedef struct _GLYPHMETRICS {
 UINT gmBlackBoxX;
 UINT gmBlackBoxY;
 POINT gmptGlyphOrigin;
 short gmCellIncX;
 short gmCellIncY;
} GLYPHMETRICS;
```

Enthält Informationen über die Plazierung und Orientierung eines Zeichenkörpers in einer Zeichenzelle, festgelegt in logischen Einheiten.

Element	Beschreibung
gmBlackBoxX	Breite des kleinsten Rechtecks, das den Zeichenkörper komplett umschließt.
gmBlackBoxY	Höhe des kleinsten Rechtecks, das den Zeichenkörper komplett umschießt.
gmptGlyphOrigin	Die x- und y-Koordinaten der oberen linken Ecke des kleinsten Rechtecks, das den Zeichenkörper komplett umschließt.
gmCellIncX	Die horizontale Entfernung des Ursprungs der aktuellen Zeichenzelle zum Ursprung der nächsten.
gmCellIncY	Die vertikale Entfernung des Ursprungs der aktuellen Zeichenzelle zum Ursprung der nächsten.

Struktur HANDLETABLE

windows.h

```
typedef struct tagHANDLETABLE {
 HGDIOBJ objectHandle[1];
} HANDLETABLE;
```

Ein Handle-Array, das eine grafische Geräteschnittstelle (GDI) identifiziert.

Struktur ICONINFO

windows.h

```
typedef struct _ICONINFO {
   BOOL    fIcon;
   DWORD   xHotspot;
   DWORD   yHotspot;
   HBITMAP hbmMask;
   HBITMAP hbmColor;
} ICONINFO;
```

Die Struktur *ICONINFO* enthält Informationen über ein Symbol oder einen Mauszeiger. Die folgende Tabelle beschreibt die Elemente.

Element	Beschreibung
fIcon	Auf TRUE gesetzt, wenn ein Symbol bezeichnet wird, und auf FALSE, wenn ein Cursor bezeichnet wird.
xHotspot	Bestimmt die x-Koordinate des Kontaktpunktes des Cursors. Für Symbole ist dies immer der Mittelpunkt; somit wird dieses Element ignoriert.
yHotspot	Bestimmt die y-Koordinate des Kontaktpunktes des Cursors. Für Symbole ist dies immer der Mittelpunkt; somit wird dieses Element ignoriert.
hbmMask	Bestimmt die Bitmaske der Symbol-Bitmap. Für schwarzweiße Symbole besteht die obere Hälfte von *hbmMask* aus einer AND-Verknüpfung des Symbols und der Bitmaske, und die untere Hälfte aus der entsprechenden XOR-Verknüpfung. Die Höhe sollte ein gerades Mehrfaches von Zwei sein. Für Farbige Symbole definiert diese Maske nur die AND Bitmaske des Symbols.
hbmColor	Bestimmt die Farb-Bitmap des Symbols. Dieses Element ist optional, wenn die Struktur ein schwarzweißes Symbol definiert. Nachdem die AND-Bitmap von *hbmMask* auf das Ziel angewendet wurde, wird die Farb-Bitmap angewendet (unter Verwendung von XOR).

Struktur KERNINGPAIR

windows.h

```
typedef struct tagKERNINGPAIR {
    WORD wFirst;
    WORD wSecond;
    int  iKernAmount;
} KERNINGPAIR;
```

Definiert ein Kerning-Paar. Die folgende Tabelle beschreibt die Elemente.

Element	Beschreibung
wFirst	Zeichencode für das erste Zeichen im Kerning-Paar.
wSecond	Zeichencode für das zweite Zeichen im Kerning-Paar.
iKernAmount	Wert, um den dieses Paar gekernt wird, wenn sie Seite an Seite im selben Font und der selben Größe erscheinen. Dieser Wert wird üblicherweise negativ sein. Er wird in logischen Einheiten angegeben.

Struktur LOGBRUSH

windows.h

```
typedef struct tagLOGBRUSH {
 UINT lbStyle;
 COLORREF lbColor;
 int lbHatch;
} LOGBRUSH;
```

Definiert die Eigenschaften, die Farbe und das Muster eines physikalischen Pinsels. Die folgende Tabelle beschreibt die Elemente.

Element	Beschreibung	
lbStyle	Legt die Eigenschaften des Pinsels fest. Dieses Element kann einen der folgenden Werte haben:	
	BS_DIBPATTERN	Muster-Pinsel, der definiert wird von einer geräteunabhängigen (DIB) Bitmap-Spezifikation.
	BS_DIBPATTERNPT	Muster-Pinsel, der definiert wird von einer geräteunabhängigen (DIB) Bitmap-Spezifikation (nur für 32.bit).
	BS_HATCHED	Schraffierter Pinsel
	BS_HOLLOW	Hohler Pinsel
	BS_PATTERN	Muster-Pinsel, der von einer Speicher-Bitmap definiert wird.
	BS_NULL	Entspricht BS_HOLLOW.
	BS_SOLID	Massiver Pinsel.
lbColor	Legt die Farbe fest, mit der der Pinsel gezeichnet wird. Wenn das Element *lbStyle* den Wert BS_HOLLOW oder BS_PATTERN hat, wird *lbColor* ignoriert. Wenn *lpStyle* den Wert BS_DIBPATTERN oder BS_DIBPATTERNPT hat, bestimmt das niederwertige Wort von *lbColor,* ob die Elemente *bmiColors* der BITMAPINFO Struktur explizite RGB-Werte enthalten oder Indizes auf die momentane logische Palette. Das Element *lbColor* muß einen der folgenden Werte haben:	
	DIB_PAL_COLORS	Farbtabelle, bestehend aus einem Array von 16-bit Indizes auf die momentane logische Palette.
	DIB_RGB_COLORS	Farbtabelle, enthält RGB-Werte.
lbHatch	Bestimmt einen Schraffurstil. Die genaue Bedeutung hängt vom Pinselstil ab. Wenn das Element *lbStyle* die Eigenschaft BS_DIBPATTERN hat, enthält das Element *lbHatch* ein Handle zu einer gepackten DIB. Wenn das Element *lbStyle* die Eigenschaft BS_DIBPATTERNPT hat, enthält das Element *lbHatch* einen Zeiger auf eine gepackte DIB. Wenn das Element lbStyle die Eigenschaft BS_HATCHED hat, bestimmt das Element lbHatch die Orientierung der Linien, aus denen die Schraffur gebildet wird. Dieses Element kann einen der	

folgenden Werte haben:

HS_BDIAGONAL	Diagonale Schraffur, aufwärts im 45-Grad Winkel (von Links nach Rechts)
HS_CROSS	Horizontale und vertikale Kreuzschraffur
HS_DIAGCROSS	45-Grad Kreuzschraffur
HS_FDIAGONAL	Diagonale Schraffur, abwärts im 45-Grad Winkel (von Links nach Rechts)
HS_HORIZONTAL	Horizontale Schraffur
HS_VERTICAL	Vertikale Schraffur

Wenn das Element *lbStyle* die Eigenschaft BS_PATTERN hat, muß *lbHatch* ein Handle zu der Bitmap sein, die das Muster definiert. Wenn das Element *lbStyle* die Eigenschaft BS_SOLID oder BS_HOLLOW hat, wird *lbHatch* ignoriert.

Siehe auch BITMAPINFO struct

Struktur LOGFONT

windows.h

```
typedef struct tagLOGFONT {
 int   lfHeight;
 int   lfWidth;
 int   lfEscapement;
 int   lfOrientation;
 int   lfWeight;
 BYTE  lfItalic;
 BYTE  lfUnderline;
 BYTE  lfStrikeOut;
 BYTE  lfCharSet;
 BYTE  lfOutPrecision;
 BYTE  lfClipPrecision;
 BYTE  lfQuality;
 BYTE  lfPitchAndFamily;
 BYTE  lfFaceName[LF_FACESIZE];
} LOGFONT;
```

32 bit Version:

```
typedef struct tagLOGFONT { /* lf */

 "pr"LONG lfHeight;
 LONG lfWidth;
 LONG lfEscapement;
 LONG lfOrientation;
 LONG lfWeight;
 BYTE lfItalic;
 BYTE lfUnderline;
 BYTE lfStrikeOut;
 BYTE lfCharSet;
 BYTE lfOutPrecision;
 BYTE lfClipPrecision;
```

```
  BYTE lfQuality;
  BYTE lfPitchAndFamily;
  CHAR lfFaceName[LF_FACESIZE];
} LOGFONT;
```

Definiert Schriftattribute. Wenn eine logische Schrift erzeugt wird, kann eine Anwendung die Vorgabewerte für die meisten Elemente benutzen. Wenn für *lfHeight* oder für *lfFaceName* keine bestimmten Werte bestimmt werden, ist die erzeugte Schrift geräteabhängig. Die folgende Tabelle beschreibt die Elemente.

Element	Beschreibung
lfHeight	(16-Bit Anwendungen) Die Höhe der Schrift, in logischen Einheiten. Ist dieser Wert kleiner als Null, bezeichnet er die Höhe der Zeichen einer Schrift. (Anwendungen, die die Höhe einer Schrift in Punkten angeben, benutzen typischerweise einen negativen Wert für dieses Element). Ist es 0, benutzt der font mapper eine voreingestellte Höhe. Der font mapper benutzt die größte physikalische Schrift, die die vorgegebene Höhe nicht überschreitet, oder die kleinste Schrift, wenn alle Schriften die gewünschte Höhe überschreiten. Der absolute Wert des Elements *lfHeight* darf 16,384 nicht überschreiten, nachdem er in gerätebezogenen Einheiten umgerechnet wurde. (32-Bit Anwendungen) Die Höhe der Schrift, in logischen Einheiten. Die Höhe der Schrift kann auf drei Wegen bestimmt werden. Wenn *lfHeight* größer Null ist, wird es in gerätebezogene Einheiten umgerechnert, und mit der Zellenhöhe der verfügbaren Schriften verglichen. Ist der Wert gleich Null, wird eine angemessene Größe vorgegeben. Ist er kleiner als Null, wird er in gerätebezogenen Einheiten umgerechnet, und der absolute Wert wird mit der Zeichenhöhe der verfügbaren Schriften verglichen. Für alle Höhenvergleiche gilt, daß der font mapper die größte Schrift heraussucht, die die gewünschte Höhe nicht überschreitet, oder, wenn eine solche Schrift nicht vorhanden ist, die kleinste vorhandene. Dieses Mapping findet statt, wenn die Schrift zum ersten Mal benutzt wird.
lfWidth	Die durchschnittliche Breite der Schriftzeichen in logischen Einheiten. Wenn dieser Wert gleich Null ist, wählt der font mapper eine voreingestellte Breite.
lfEscapement	Der Winkel zwischen der Grundlinie eines Zeichens und seiner x-Achse in Zehntelgraden.
lfOrientation	Der Winkel zwischen der Grundline eines Zeichens und der Grundlinie der Seite, in Zehntelgraden.
lfWeight	Strichstärke. Kann einen der folgenden Werte haben: FW_DONTCARE — FW_SEMIBOLD FW_THIN — FW_DEMIBOLD FW_EXTRALIGHT — FW_BOLD FW_ULTRALIGHT — FW_BOLD FW_LIGHT — FW_ULTRABOLD FW_NORMAL — FW_BLACK FW_REGULAR — FW_HEAVY FW_MEDIUM Ist der Wert 0, wird eine voreingestellte Strichstärke benutzt.
lfItalic	Kursiv wenn TRUE.
lfUnderline	Unterstrichen wenn TRUE.
lfStrikeOut	Durchgestrichen wenn TRUE.

lfCharSet — Zeichensatz. Kann einen der folgenden Werte haben:

ANSI_CHARSET
DEFAULT_CHARSET (16 Bit)
SYMBOL_CHARSET
SHIFTJIS_CHARSET (16 Bit)
OEM_CHARSET
UNICODE_CHARSET (32 Bit)

lfOutPrecision — Gewünschte Ausgabegenauigkeit. Legt fest, wie genau die Ausgabe mit der verlangten Schrift in Höhe, Breite, Orientierung an x-Achse und Grundlinie und Schreibdichte angestimmt sein muß. Kann einen der folgenden Werte haben:

OUT_CHARACTER_PRECIS OUT_STRING_PRECIS
OUT_DEFAULT_PRECIS OUT_STROKE_PRECIS
OUT_DEVICE_PRECIS OUT_TT_PRECIS
OUT_RASTER_PRECIS OUT_TT_ONLY_PRECIS

lfClipPrecision — Gewünschte Schnittgenauigkeit. Legt fest, wie Zeichen abgeschnitten werden, die sich teilweise außerhalb des Schnittbereiches befinden. Kann einen der folgenden Werte haben:

CLIP_CHARACTER_PRECIS CLIP_MASK
CLIP_DEFAULT_PRECIS CLIP_STROKE_PRECIS
CLIP_EMBEDDED CLIP_TT_ALWAYS
CLIP_LH_ANGLES

Um eine eingebettete Schrift, die nur gelesen werden kann, zu benutzen, muß eine Anwendung den Wert CLIP_EMBEDDED festlegen.

Um eine gleichmäßige Drehung von geräteabhängigen Schriften, TrueType-Schriften und Vektorschriften zu erreichen, kann eine Anwendung den Operator OR benutzen, um den Wert CLIP_LH_ANGLES mit jedem der anderen *lfClipPrecision* Werte zu kombinieren. Wenn das Bit CLIP_LH_ANGLES gesetzt ist, hängt die Drehung aller Schriften davon ab, ob die Orientierung des Koordinatensystems rechts- oder linksdrehend ist. Wenn CLIP_LH_ANGLES nicht definiert ist, werden geräteabhängige Schriften immer gegen den Uhrzeigersinn gedreht. Die Drehung der anderen Schriften allerdings hängt von der Orientierung des Koordinatensystems ab. (Für mehr Informationen über die Orientierung von Koordinatensystemen siehe auch die Beschreibung des Elements *lfEscapement*.)

lfQuality — Qualität der Ausgabe der Schrift. Kann einen der folgenden Werte haben:

Wert	**Bedeutung**
DEFAULT_QUALITY	Das Aussehen der Schrift ist nicht von Bedeutung.
DRAFT_QUALITY	Dem Aussehen der Schrift wird weniger Bedeutung zugemessen, als wenn der Wert PROOF_QUALITY benutzt würde.
PROOF_QUALITY	Die Qualität der Ausgabe der Schrift ist wichtiger als die Beachtung der logischen Schriftattribute.

lfPitchAndFamily Schriftfamilie und Schreibdichte. Die beiden niederwertigen Bits legen die Schreibdichte fest und können einen der folgenden Werte haben:

DEFAULT_PITCH
FIXED_PITCH
VARIABLE_PITCH

Die vier höherwertigen Bits des Elements bestimmen die Schriftfamilie und können einen der folgenden Werte haben:

Wert	**Bedeutung**
FF_DECORATIVE	Zierschriften.
FF_DONTCARE	Nicht bekannt oder nicht relevant.
FF_MODERN	Schriften mit festem Zeichenabstand, mit oder ohne Serifen.

	(z.B. Pica und Courier).
FF_ROMAN	Schriften mit variablem Zeichenabstand und mit Serifen (z.B. Times New Roman).
FF_SCRIPT	Schriften, die wie Handschriften aussehen (z.B. Script und Kursiv).
FF_SWISS	Schriften mit variablem Zeichenabstand und ohne Serifen (z.B. Sans Serif).

Benutzen Sie den booleschen Operator OR um eine Schreibdichtenkonstante mit einer Schriftfamilienkonstante zu vergleichen.

lfFaceName — Legt den Namen der Schrift fest. Die Länge des Strings darf LF_FACESIZE - 1 nicht überschreiten.

Struktur LOGPALETTE

windows.h

```
typedef struct tagLOGPALETTE {
 WORD palVersion;
 WORD palNumEntries;
 PALETTEENTRY palPalEntry[1];
} LOGPALETTE;
```

Definiert eine logische Farbpalette. Die Farben in der Entry-Tabelle der Palette sollten in der Reihenfolge ihrer Wichtigkeit aufgeführt sein, da ein Eintrag in der logischen Palette um so wahrscheinlicher in die Systempalette aufgenommen wird, je eher er in der logischen Palette erscheint. Die folgende Tabelle beschreibt die Elemente:

Element	Beschreibung
palVersion	Legt die Windows Versionsnummer für die Struktur fest(momentan 0x300).
palNumEntries	Legt die Anzahl der Einträge in der logischen Farbpalette fest.
palPalEntry	Legt einen Array von *PALETTEENTRY* Strukturen fest, der die Farbe und die Verwendung eines jeden Eintrags in der logischen Palette bestimmt.

Struktur LOGPEN

windows.h

```
typedef struct tagLOGPEN {
 UINT lopnStyle;
 POINT lopnWidth;
 COLORREF lopnColor;
} LOGPEN;
```

Definiert die Eigenschaften, die Breite und die Farbe eines Stifts.

Element	Beschreibung
lopnStyle	Bestimmt die Art der gezeichneten Linie. Dieses Element kann einen der folgenden Werte haben: Wert — Bedeutung PS_SOLID — Duchgehende Linie PS_DASH — Gestrichelte Linie (Stiftbreite = 1) PS_DOT — Gepunktete Linie (Stiftbreite = 1) PS_DASHDOT — Alternierend gestrichelte und gepunktete Linie (Stiftbreite = 1) PS_DASHDOTDOT — Alternierend gestrichelte und doppelt gepunktete Linie (Stiftbreite = 1) PS_NULL — Unsichtbare Linie PS_INSIDEFRAME — Erstellt einen Stift, der eine Linie innerhalb geschlossener Formen zeichnet, die von der Ausgabefunktion der grafischen Geräteschnittstelle (GDI) erzeugt werden, die ein begrenzendes Rechteck bestimmt. Wenn dieser Stil mit GDI Ausgabefunktionen gebraucht wird.
lopnWidth	Legt die Breite des Zeichenwerkzeugs in logischen Einheiten fest. Wenn das Element *lopnWidth* gleich Null ist, ist die gezeichnete Linie genau einen Pixel breit, unabhängig vom momentanen mapping Modus.
lopnColor	Bestimmt die Farbe des Zeichenwerkzeugs.

Struktur MDICREATESTRUCT

```
typedef struct tagMDICREATESTRUCT {
 LPCSTR    szClass;
 LPCSTR    szTitle;
 HANDLE hOwner;
 int       x;
 int       y;
 int       cx;
 int       cy;
 DWORD     style;
 LPARAM    lParam;
} MDICREATESTRUCT;
```

Enthält Klasse, Titel, Ort, Besitzer und Größe des Fensters einer untergeordneten Multidokumentschnittstelle (MDI). Die untenstehende Tabelle beschreibt die Elemente.

Element	Beschreibung
szClass	Zeigt auf einen nullterminierten String, der den registrierten Namen des untergeordneten MDI-Fensters festlegt.
szTitle	Zeigt auf einen nullterminierten String, der den registrierten Titel des untergeordneten MDI-Fensters festlegt.

hOwner	Instanz-Handle für die Anwendung, die das untergeordnete MDI-Fenster erstellt.
x	Anfangsposition der linken Seite des untergeordneten MDI-Fensters. Ist dieser Wert auf CW_USEDEFAULT gesetzt, wird dem untergeordneten MDI-Fenster eine horizontale Position zugewiesen.
y	Anfangsposition der oberen Seite des untergeordneten MDI-Fensters. Ist dieser Wert auf CW_USEDEFAULT gesetzt, wird dem untergeordneten MDI-Fenster eine vertikale Position zugewiesen.
cx	Anfangsbreite des untergeordneten MDI-Fensters. Ist dieser Wert auf CW_USEDEFAULT gesetzt, wird dem untergeordneten MDI-Fenster eine voreingestellte Breite zugewiesen.
cy	Anfangshöhe des untergeordneten MDI-Fensters. Ist dieser Wert auf CW_USEDEFAULT gesetzt, wird dem untergeordneten MDI-Fenster eine voreingestellte Höhe zugewiesen.
style	Legt weitere Eigenschaften des untergeordneten MDI-Fensters fest. Wenn das MDI-Client-Fenster mit der Fensterart MDIS_ALLCHILDSTYLES erstellt wurde, kann diese Element jede Kombination von Fenstereigenschaften und Erstellungsattributen sein, einschließlich der folgenden:
	Wert — **Bedeutung**
	WS_MINIMIZE — Als Symbol erstellt.
	WS_MAXIMIZE — Als Vollbild erstellt.
	WS_HSCROLL — Mit einer horizontalen Bildlaufleiste erstellt.
	WS_VSCROLL — Mit einer vertikalen Bildlaufleiste erstellt.
lParam	Ein von der Anwendung definierter 32-Bit Wert.

Struktur METARECORD

windows.h

```
typedef struct tagMETARECORD {
 DWORD rdSize;
 WORD rdFunction;
 WORD rdParm[1];
} METARECORD;
```

Enthält einen Metadatei-Datensatz. Die untenstehende Tabelle beschreibt die Elemente.

Element	Beschreibung
rdSize	Größe des Datensatzes, in Worten.
rdFunction	Funktionsnummer.
rdParm	Funktionsparameter. Werden in umgekehrter Reihenfolge an die Funktion weitergegeben.

Struktur MEASUREITEMSTRUCT

windows.h

```
typedef struct tagMEASUREITEMSTRUCT {
 UINT CtlType;
```

```
 UINT CtlID;
 UINT itemID;
 UINT itemWidth;
 UINT itemHeight;
 DWORD itemData;
} MEASUREITEMSTRUCT;
```

Enthält die Dimensionen eines vom Besitzer gezeichneten Dialogelements. Die untenstehende Tabelle beschreibt die Elemente:

Element	Beschreibung
CtlType	Legt den Kontrolltyp fest und kann einen der folgenden Werte haben: ODT_BUTTON V om Besitzer gezeichnete Schaltfläche ODT_COMBOBOX Vom Besitzer gezeichnetes Kombinationsfenster ODT_LISTBOX Vom Besitzer gezeichnetes Listenfeld ODT_MENU Vom Besitzer gezeichnetes Menü
CtlID	Bezeichner des Kombinationsfensters, des Listenfeldes oder der Schaltfläche (wird nicht für Menüs benutzt)
itemID	Bezeichner eines Menüeintrags für einen Menüeintrag, oder Bezeichner eines Listenfeldes für eine Liste oder ein Kombinationsfenster variabler Höhe (wird nicht benutzt für Listen und Kombinationsfenster mit fester Höhe oder Schaltflächen).
itemWidth	Breite eines Menüelements, in Pixeln.
itemHeight	Höhe eines individuellen Eintrags in ein Listenfeld oder ein Menü, in Pixeln.
itemData	Anwendungsdefinierter 32-Bit Wert, der zu einem Menüeintrag gehört. Für ein Steuerelement bestimmt dieser Parameter den letzten Wert, der dem Listenfeld oder dem Kombinationsfenster von der Meldung LB_SETITEMDATA oder CB_SETITEMDATA zugewiesen wurde. Wenn das Listenfeld die Eigenschaften LB_HASSTRINGS oder CB_HASSTRINGS hat, ist der Anfagswert Null. Andernfalls ist dieser Anfangswert derjenige, der mit dem Parameter *lParam* an das Listenfeld oder das Kombinationsfenster in einer der folgenden Meldungen übergeben wurde: CB_ADDSTRING CB_INSERTSTRING LB_ADDSTRING LB_INSERTSTRING

Struktur MSG

windows.h

```
typedef struct tagMSG {
 HWND hwnd;
 UINT message;
 WPARAM wParam;
 LPARAM lParam;
 DWORD time;
 POINT pt;
} MSG;
```

Die Struktur *MSG* enthält die folgenden Meldungsinformationen aus einer Meldungswarteschlange:

Element	Beschreibung
hwnd	Handle für das Fenster, dessen Fensterprozedur die Meldung erhält.
message	Meldungsnummer.
wParam	Zusätzliche Meldungsinformationen. Die Bedeutung hängt vom Wert von *message* ab.
lParam	Zusätzliche Meldungsinformationen. Die Bedeutung hängt vom Wert von *message* ab.
time	Der Zeitpunkt, an dem die Meldung abgeschickt wurde.
pt	Die Cursorposition zu dem Zeitpunkt als die Meldung abgeschickt wurde, in Bildschirmkoordinaten.

Struktur OUTLINETEXTMETRIC

windows.h

```
typedef struct _OUTLINETEXTMETRIC {
 UINT   otmSize;
 TEXTMETRIC otmTextMetrics;
 BYTE   otmFiller;
 PANOSE otmPanoseNumber;
 UINT   otmfsSelection;
 UINT   otmfsType;
 int    otmsCharSlopeRise;
 int    otmsCharSlopeRun;
 int    otmItalicAngle;
 UINT   otmEMSquare;
 int    otmAscent;
 int    otmDescent;
 UINT   otmLineGap;
 UINT   otmsCapEmHeight;
 UINT   otmsXHeight;
 RECT   otmrcFontBox;
 int    otmMacAscent;
 int    otmMacDescent;
 UINT   otmMacLineGap;
 UINT   otmusMinimumPPEM;
 POINT  otmptSubscriptSize;
 POINT  otmptSubscriptOffset;
 POINT  otmptSuperscriptSize;
 POINT  otmptSuperscriptOffset;
 UINT   otmsStrikeoutSize;
 int    otmsStrikeoutPosition;
 int    otmsUnderscoreSize;
 int    otmsUnderscorePosition;
 PSTR   otmpFamilyName;
 PSTR   otmpFaceName;
 PSTR   otmpStyleName;
 PSTR   otmpFullName;
} OUTLINETEXTMETRIC;
```

Enthält die metrische Beschreibung einer TrueType Schrift. Die Größen werden zurückgegeben in logischen Einheiten (abhängig vom momentanen Mapping-Modus der Anzeige). Die folgende Tabelle beschreibt die Elemente:

Element	Beschreibung
otmSize	Größe der Struktur, in Bytes.
otmTextMetrics	TEXTMETRIC Struktur, enthält zusätzliche Schriftinformationen.
otmFiller	Wert, der diese Struktur byteweise ausrichtet.
otmPanoseNumber	Die PANOSE Zahl für diese Schrift.
otmfsSelection	Art des Schriftmusters. Kann eine Kombination folgender Bits sein: Bit Bedeutung 0 Kursiv 1 Unterstrichen 2 Negativ 3 Konturschrift 4 Durchgestrichen 5 Fett
otmfsType	Bestimmt, ob eine Schrift lizensiert ist. Lizensierte Schriften dürfen weder verändert noch ausgetauscht werden. Wenn das Bit 1 gesetzt ist, kann die Schrift nicht in ein Dokument eingebettet werden. Ist Bit 1 nicht gesetzt, kann die Schrift eingebettet werden. Wenn Bit 2 gesetzt ist, kann die eingebettete Schrift nur gelesen werden.
otmsCharSlopeRise	Neigung des Mauszeigers. Bei einer vertikalen Neigung ist dieser Wert 1. Anwendungen können diesen Wert und den Wert des Elements *otmsCharSlopeRun* dazu benutzen, einen kursiven Mauszeiger zu erstellen, der den selben Neigungswinkel hat wie die Kursivschrift. (Wird vom Element *otmItalicAngle* bestimmt).
otmsCharSlopeRun	Neigung des Mauszeigers. Bei einer vertikalen Neigung ist dieser Wert 0. Anwendungen können diesen Wert und den Wert des Elements *otmsCharSlopeRise* dazu benutzen, einen kursiven Mauszeiger zu erstellen, der den selben Neigungswinkel hat wie die Kursivschrift. (Wird vom Element *otmItalicAngle* bestimmt).
otmItalicAngle	Neigungswinkel der Kursivschrift in Graden, gerechnet gegen den Uhrzeigersinn von der vertikalen Achse aus gemessen. Normale (Romanische) Schriften haben einen Neigungswinkel mit dem Wert Null. Kursivschriften haben üblicherweise einen negativen Neigungswinkel (d.h. sie lehnen sich nach rechts).
otmEMSquare	Anzahl der logischen Einheiten, die die x- oder y- Dimension des em-Quadrats einer Schrift definieren. (Die Anzahl der Einheiten in der x- und y- Richtung ist immer dieselbe wie für ein em-Quadrat).
otmAscent	Maximale Entfernung, die die Zeichen dieser Schrift oberhalb der Grundlinie erreichen. Dies ist die typograhpische Oberlänge dieser Schrift.
otmDescent	Maximale Entfernung, die die Zeichen dieser Schrift unterhalb der Grundlinie erreichen. Dies ist die typographische Unterlänge der Schrift.
otmLineGap	Typographischer Zeilenabstand.
otmsCapEmHeight	Nicht unterstützt.
otmsXHeight	Nicht unterstützt.
otmrcFontBox	Begrenzungsrahmen für die Schrift.

otmMacAscent	Maximale Entfernung, die die Zeichen dieser Schrift oberhalb der Grundlinie erreichen für Macintosh Computer.
otmMacDescent	Maximale Entfernung, die die Zeichen dieser Schrift unterhalb der Grundlinie erreichen für Macintosh Computer.
otmMacLineGap	Zeilenabstand für Macintosh Computer.
otmusMinimumPPEM	Legt die kleinste empfohlene Größe für diese Schrift fest in Pixel pro em-Quadrat.
otmptSubscriptSize	Empfohlene horizontale und vertikale Größe für tiefergestellte Schrift in dieser Schrift.
otmptSubscriptOffset	Empfohlener horizontaler und vertikaler Offset für tiefergestellte Zeichen in dieser Schrift. Der Offset für tiefergestellte Zeichen in dieser Schrift wird gemessen vom Ursprung des Zeichens bis zum Ursprung des tiefergestellten Zeichens.
otmptSuperscriptSize	Empfohlene horizontale und vertikale Größe für hochgestellte Zeichen in dieser Schrift.
otmptSuperscriptOffset	Empfohlener horizontaler und vertikaler Offset für tiefergestellte Zeichen in dieser Schrift. Der Offset für hochgestellte Schrift in dieser Schrift wird gemessen von der Grundlinie des Zeichens bis zur Grundlinie des hochgestellten Zeichens.
otmsStrikeoutSize	Breite des strikeout stroke für diese Schrift. Typischerweise ist dies die Breite des em-Strichs für diese Schrift.
otmsStrikeoutPosition	Position des strikeout stroke relativ zur Grundlinie dieser Schrift. Positive Werte liegen über der Grundlinie, negative darunter.
otmsUnderscoreSize	Dicke des Unterstriches für diese Schrift.
otmsUnderscorePosition	Position des Unterstriches für diese Schrift.
otmpFamilyName	Offset vom Anfang der Struktur zu einem String, der den Familiennamen der Schrift festlegt.
otmpFaceName	Offset vom Anfang der Struktur zu einem String, in dem die Schriftart der Schrift festgelegt wird. (Dieser Schriftname entspricht dem Namen, der in der LOGFONT Struktur festgelegt ist.)
otmpStyleName	Offset vom Anfang der Struktur zu einem String, in dem der Stilname der Schrift festgelegt wird.
otmpFullName	Offset vom Anfang der Struktur zu einem String, in dem der vollständige Name der Schrift festgelegt wird. Dieser Name ist ausschließlich dieser Schrift vorbehalten, und enthält oft eine Versionsnummer oder andere identifizierende Informationen.

Struktur RGBQUAD

windows.h

```
typedef struct tagRGBQUAD {
 BYTE rgbBlue;
 BYTE rgbGreen;
 BYTE rgbRed;
 BYTE rgbReserved;
} RGBQUAD;
```

Beschreibt eine Farbe, die duch unterschiedliche Intensität von Rot, Grün und Blau gebildet wird. Die folgende Tabelle beschreibt die Elemente.

Element	Beschreibung
rgbBlue	Intensität des Blau
rgbGreen	Intensität an Grün
rgbRed	Intensität an Rot
rgbReserved	Muß Null sein.

Struktur RGBTRIPLE

windows.h

```
typedef struct tagRGBTRIPLE {
 BYTE rgbtBlue;
 BYTE rgbtGreen;
 BYTE rgbtRed;
} RGBTRIPLE;
```

Beschreibt eine Farbe, die duch unterschiedliche Intensität von Rot, Grün und Blau gebildet wird.

Element	Beschreibung
rgbtBlue	Intensität an Blau
rgbtGreen	Intensität an Grün
rgbtRed	Intensität an Rot

Struktur TEXTMETRIC

windows.h

Die 16-Bit Version wird anders formatiert und gepackt als die 32-Bit Version:

```
typedef struct tagTEXTMETRIC {
 int  tmHeight;
 int  tmAscent;
 int  tmDescent;
 int  tmInternalLeading;
 int  tmExternalLeading;
 int  tmAveCharWidth;
 int  tmMaxCharWidth;
 int  tmWeight;
 BYTE tmItalic;
 BYTE tmUnderlined;
 BYTE tmStruckOut;
 BYTE tmFirstChar;
 BYTE tmLastChar;
 BYTE tmDefaultChar;
 BYTE tmBreakChar;
 BYTE tmPitchAndFamily;
 BYTE tmCharSet;
 int  tmOverhang;
```

```
  int  tmDigitizedAspectX;
  int  tmDigitizedAspectY;
} TEXTMETRIC;
```

32-bit version:

```
typedef struct tagTEXTMETRIC {
  LONG tmHeight;
  LONG tmAscent;
  LONG tmDescent;
  LONG tmInternalLeading;
  LONG tmExternalLeading;
  LONG tmAveCharWidth;
  LONG tmMaxCharWidth;
  LONG tmWeight;
  LONG tmOverhang;
  LONG tmDigitizedAspectX;
  LONG tmDigitizedAspectY;
  BYTE tmFirstChar;
  BYTE tmLastChar;
  BYTE tmDefaultChar;
  BYTE tmBreakChar;
  BYTE tmItalic;
  BYTE tmUnderlined;
  BYTE tmStruckOut;
  BYTE tmPitchAndFamily;
  BYTE tmCharSet;
} TEXTMETRIC;
```

Enthält grundlegende Informationen über eine physikalische Schrift. Alle Größen sind logische Einheiten (Abhängig vom Mapping-Modus). Die untenstehende Tabelle beschreibt die Elemente.

Elemente	Beschreibung
tmHeight	Höhe der Zeichenzellen. (*tmAscent* + *tmDescent*)
tmAscent	Oberlänge der Zeichenzellen. (in Einheiten oberhalb der Grundlinie)
tmDescent	Unterlänge der Zeichenzellen. (in Einheiten unterhalb der Grundlinie)
tmInternalLeading	Anzahl der führenden (Leerzeichen) innerhalb der Begrenzung, die duch *tmHeight* bestimmt wird. Diakritische Zeichen kommen in diesem Bereich vor.
tmExternalLeading	Anzahl zusätzlicher führender (Leerzeichen), die zwischen den Zeilen hinzugefügt werden. Manchmal auf Null gesetzt.
tmAveCharWidth	Duchschnittliche Zeichenbreite der Schrift ohne Unterschneidungen.
tmMaxCharWidth	Breite des breitesten Zeichens der Schriftart.
tmWeight	Strichstärke. Kann einen der folgenden Werte haben: FW_DONTCARE FW_SEMIBOLD FW_THIN FW_DEMIBOLD FW_EXTRALIGHT FW_BOLD FW_ULTRALIGHT FW_EXTRABOLD FW_LIGHT FW_ULTRABOLD FW_NORMAL FW_BLACK

	FW_REGULAR	FW_HEAVY
	FW_MEDIUM	
tmItalic	Kursiv wenn nicht 0.	
tmUnderlined	Unterstrichen wenn nicht 0.	
tmStruckOut	Durchgestrichen wenn nicht 0.	
tmFirstChar	Wert des ersten Zeichens, das von der Schrift definiert wird.	
tmLastChar	Wert des letzen Zeichens, das von der Schrift definiert wird.	
tmDefaultChar	Wert des Zeichens, das nicht im Zeichensatz enthaltene Zeichen ersetzt.	
tmBreakChar	Trennzeichen-Wert für Ausrichtung des Textes.	
tmPitchAndFamily	Schreibdichte und Familie der ausgewählen Schrift. Die vier niederwertigen Bit identifizieren den Schrifttyp wie folgt:	
	TMPF_FIXED_PITCH	Schrift mit fester Schreibdichte.
	TMPF_VECTOR	Vektor oder TrueType Schrift.
	TMPF_TRUETYPE	TrueType Schrift.
	TMPF_DEVICE	Geräteschrift
	Die vier höherwertigen Bits dieses Elements bezeichenen die Schriftfamilie. Das Element *tmPitchAndFamily* kann mit dem Hexadezimalen Wert 0xF0 kombiniert werden, indem man den bitweisen Operator AND benutzt. Dann können die Namen der Schriftfamilien verglichen werden, bis eine Übereinstimmung erzielt wurde. Die folgenden Schriftfamilien sind definiert:	
	Wert	**Bedeutung**
	FF_DECORATIVE	Zierschriften
	FF_DONTCARE	Nicht bekannt oder nicht relevant
	FF_MODERN	Schriften mit festem Zeichenabstand, mit oder ohne Serifen.
	FF_ROMAN	Schriften mit variablem Zeichenabstand und mit Serifen.
	FF_SCRIPT	Schriften, die wie Handschrift wirken.
	FF_SWISS	Schriften mit variablem Zeichenabstand und ohne Serifen.
tmCharSet	Bestimmt den Zeichensatz der Schrift. Die folgenden Werte sind definiert:	
	ANSI_CHARSET	
	DEFAULT_CHARSET	
	SYMBOL_CHARSET	
	SHIFTJIS_CHARSET	
	OEM_CHARSET	
tmOverhang	Zusätzliche Breite, die zu einigen synthetisierten Schriften hinzugefügt wird.	
tmDigitizedAspectX	Horizontaler Aspekt des Gerätes, für das die Schrift erstellt wurde. Das Verhältnis von *tmDigitizedAspectX* und *tmDigitizedAspectY* ist das XY-Verhältnis des Gerätes, für das diese Schrift erstellt wurde.	
tmDigitizedAspectY	Vertikaler Aspekt des Gerätes, für das die Schrift erstellt wurde. Das Verhältnis von *tmDigitizedAspectX* und *tmDigitizedAspectY* ist das XY-Verhältnis des Gerätes, für das diese Schrift erstellt wurde.	

WNDCLASS struct

windows.h

```
typedef struct tagWNDCLASS {
 UINT    style;
 WNDPROC lpfnWndProc;
 int     cbClsExtra;
```

```
  int     cbWndExtra;
  HANDLE  hInstance;
  HICON   hIcon;
  HCURSOR hCursor;
  HBRUSH  hbrBackground;
  LPCTSTR lpszMenuName;
  LPCTSTR lpszClassName;
} WNDCLASS;
```

Enthält Klassenregistrationsattribute für *TWindows*, die in der folgenden Tabelle beschrieben werden:

Element	Beschreibung
style	Eigenschaften der Klasse. Kann jegliche Kombination folgender Werte sein: Wert — Bedeutung CS_BYTEALIGNCLIENT — Richtet den Client-Bereich des Fensters an der Byte-Begrenzung aus (in x-Richtung). CS_BYTEALIGNWINDOW — Richtet ein Fenster an einer Byte-Begrenzung (in x-Richtung) aus, um die Geschwindigkeit bei Operationen zu erhöhen, die Bewegung oder Größenveränderungen des Fensters beinhalten. Diese Eigenschaft beeinflußt die Breite eines Fensters und die horizontale Position auf dem Bildschirm. CS_CLASSDC — Weist allen Fenstern einer Klasse einen Gerätekontext (DC) zum gemeinsamen Zugriff zu. CS_DBLCLKS — Sendet eine Doppelklick-Meldung an die Fensterprozedur, wenn sich der Mauszeiger innerhalb eines Fensters befindet, das zu dieser Klasse gehört. CS_GLOBALCLASS — Erlaubt es einer Anwendung, ein Fenster dieser Klasse zu erstellen, unabhängig vom Wert des Parameters *hInstance*, der bei der Erstellung des Fensters benutzt wird.
lpfnWndProc	Zeigt auf die Fensterprozedur.
cbClsExtra	Anzahl zusätzlicher zuzuweisender Bytes, entsprechend der Fensterklassenstruktur. Das Betriebssystem initialisiert die Bytes zu Null.
cbWndExtra	Anzahl zusätzlicher zuzuweisender Bytes, entsprechend der Fensterinstanz. Wird zu Null initialisiert.
hInstance	Identifiziert die Instanz, innerhalb derer sich die Fensterprozedur dieser Klasse befindet.
hIcon	Identifiziert das Klassensymbol. Muß ein Handle einer Symbolressource sein. Wenn es NULL ist, muß eine Anwendung ein Symbol zeichnen, wann immer der Benutzer das Anwendungsfenster minimiert.
hCursor	Identifiziert den Klassenmauszeiger. Muß ein Handle einer Mauszeiger-Ressource sein. Wenn es NULL ist, muß eine Anwendung immer dann die Mauszeigerform zur Verfügung stellen, wenn sich die Maus in das Anwendungsfenster bewegt.
hbrBackground	Identifiziert den Klassenpinsel. Kann ein Handle auf einen physikalischen Pinsel sein, um den Hintergrund zu zeichnen, oder ein Farbwert. Ein Farbwert muß eine der Standard-Systemfarben sein, die unten aufgeführt werden, plus 1. Wenn ein Farbwert gegeben ist, müssen Sie diesen in einen der folgenden HBRUSH Werte umwandeln. COLOR_ACTIVEBORDER — COLOR_HIGHLIGHTTEXT COLOR_ACTIVECAPTION — COLOR_INACTIVEBORDER COLOR_APPWORKSPACE — COLOR_INACTIVECAPTION COLOR_BACKGROUND — COLOR_MENU

	COLOR_BTNFACE COLOR_MENUTEXT COLOR_BTNSHADOW COLOR_SCROLLBAR COLOR_BTNTEXT COLOR_WINDOW COLOR_CAPTIONTEXT COLOR_WINDOWFRAME COLOR_GRAYTEXT COLOR_WINDOWTEXT COLOR_HIGHLIGHT Keine Anwendung sollte diese Pinsel löschen, da eine Klasse von mehreren Instanzen einer Anwendung benutzt werden kann. Wenn dieses Element NULL ist, muß eine Anwendung Ihren eigenen Hintergrund zeichnen, wann immer es nötig ist, in seinem Client-Bereich zu zeichnen.
lpszMenuName	Zeigt auf einen nullterminierten Zeichenstring, der den Ressourcennamen des Klassenmenüs bestimmt, so wie der Name in der Ressourcendatei erscheint. Ist dieser Wert NULL, haben Fenster, die zu dieser Klasse gehören, keine vorgegebenen Menüs.
lpszClassName	Zeigt auf einen nullterminierten String, oder ist ein Atom, das den Fensterklassennamen bestimmt.

Struktur PAINTSTRUCT

windows.h

16-bit Version:

```
typedef struct tagPAINTSTRUCT {
 HDC  hdc;
 BOOL fErase;
 RECT rcPaint;
 BOOL fRestore;
 BOOL fIncUpdate;
 BYTE rgbReserved[16];
} PAINTSTRUCT;
```

32-bit version:

```
typedef struct tagPAINTSTRUCT {
 "pr"HDC  hdc;
 BOOL fErase;
 RECT rcPaint;
 BOOL fRestore;
 BOOL fIncUpdate;
 BYTE rgbReserved[32];
} PAINTSTRUCT;
```

Enthält Informationen, die eine Anwendung benötigt, um im Client-Bereich eines Fensters zu zeichnen.

Element	Beschreibung
hdc	Der zum Zeichnen benutzte Bildschirmkontext.
fErase	Wenn nicht-Null, den Hintergrund löschen.
rcPaint	Bestimmt die obere linke und die untere rechte Ecke des zu zeichnenden Rechtecks.

fRestore	Reserviert
fIncUpdate	Reserviert
rgbReserved	Reserviert

Struktur PALETTEENTRY

windows.h

```
typedef struct tagPALETTEENTRY {
 BYTE peRed;
 BYTE peGreen;
 BYTE peBlue;
 BYTE peFlags;
} PALETTEENTRY;
```

Bestimmt die Farbe und den Gebrauch eines Eintrags in eine logische Farbpalette (Siehe auch *LOGPALETTE* struct) Die folgende Tabelle beschreibt die Einstellungen:

Element	Beschreibung
peRed	Bestimmt einen Rot-Intensitätswert für den Paletteneintrag.
peGreen	Bestimmt einen Grün-Intensitätswert für den Paletteneintrag.
peBlue	Bestimmt einen Blau-Intensitätswert für den Paletteneintrag.
peFlags	Bestimmt, wie ein Paletteneintrag benutzt wird. Das Element *peFlags* kann entweder NULL sein, oder einen der folgenden Werte haben:

Wert	Beschreibung
PC_EXPLICIT	Das niederwertige Wort der logischen Palette bezeichnet einen Hardware-Palettenindex. Dieses Flag erlaubt es der Anwendung, den Inhalt der Bildschirmpalette anzuzeigen.
PC_NOCOLLAPSE	Die Farbe steht in einem unbenutzten Eintrag der Systempalette, anstatt mit einer bereits vorhandenen Systempalettenfarbe abgeglichen zu werden.
PC_RESERVED	Bestimmt, daß der logische Paletteneintrag zur Animation der Palette benutzt wird, und verhindert dadurch, daß andere Fenster an den Paletteneintrag Farben anpassen, da sich die Farbe ständig ändert. Wenn ein unbenutzter Systempaletteneintrag vorhanden ist, wird die Farbe dort eingetragen. Andernfalls steht die Farbe für Animationen zur Verfügung.

Struktur XFORM

windows.h

```
typedef struct  tagXFORM {
 FLOAT eM11;
```

```
  FLOAT eM12;
  FLOAT eM21;
  FLOAT eM22;
  FLOAT eDx;
  FLOAT eDy;
} XFORM;
```

Gibt eine world-space zu page-space Transformation an. Die folgende Tabelle beschreibt die Elemente.

Element	Beschreibung
eM11	Legt folgendes fest:
Wert	Bedeutung
Scaling	Horizontale Skalierungskomponente.
Rotation	Cosinus des Rotationswinkels
Reflection	Horizontale Komponente
eM12	Legt folgendes fest:
Wert	Bedeutung
Shear	Horizontale proportionale Komponente
Rotation	Sinus des Rotationswinkels
eM21	Legt folgendes fest:
Wert	Bedeutung
Shear	Vertikale proportionale Komponente
Rotation	Negativer Sinus des Rotationswinkels
eM22	Legt folgendes Fest:
Wert	Bedeutung
Scaling	Vertikale Skalierungskomponente
Rotation	Coninus des Rotationswinkels
Reflection	Vertikale Rotationskomponente
eDx	Bestimmt die horizontale Übersetzungskomponente.
eDy	Bestimmt die vertikale Übersetzungskomponente.

Die folgende Tabelle zeigt, wie die Elemente für jede Operation genutzt werden können.

A N H A N G C

Vererbungsdiagramme

Dieses Kapitel enthält die ObjectWindows-Vererbungsdiagramme in alphabetischer Reihenfolge. Mit Hilfe dieser Diagramme können Sie Funktionen nachsehen, die für eine einzelne Klasse neu oder redefiniert sind. Sie können ermitteln, welche virtuellen Funktionen von abgeleiteten Klassen geerbt sind und welche virtuellen Funktionen in der am weitesten abgeleiteten Klasse deklariert sind.

Funktionen, die in der abgeleiteten Klasse als virtuell definiert sind, werden unterstrichen. Funktionen, die in der übergeordneten Klasse deklariert und in der abgeleiteten Klasse überschrieben sind, werden in der übergeordneten Klasse kursiv dargestellt. Nach Definition sind diese kursiven Funktionen ebenso als virtuell definiert. Zum Beispiel überschreibt die virtuelle Funktion *TGadgetWindow::Paint* die virtuelle Funktion *TWindow::Paint*.

TApplication applicat.h

TModule

AccessResource	GetModuleUsage	LoadResource
AllocResource	GetName	LoadString
CopyCursor	GetParentObject	LowMemory
CopyIcon	GetProcAddress	MakeWindow
Error	InitModule	RestoreMemory
ExecDialog	IsLoaded	SetInstance
FindResource	LoadAccelerators	SetResourceHandler
GetClassInfo	LoadBitmap	SizeOfResource
GetInstance	LoadCursor	ValidWindow
GetInstanceData	LoadIcon	
GetModuleFileName	LoadMenu	

TApplication

BeginModal	GetWinParams	ResumeThrow
BWCCEnabled	IdleAction	Run
CanClose	InitApplication	SetDocManager
Condemn	InitInstance	SetMainWindow
Ctl3dEnabled	InitMainWindow	SetMainWinParams
EnableBWCC	MessageLoop	SuspendThrow
EnableCtl3dAutosubclass	PostDispatchAction	TermInstance
EndModal	PreProcessMenu	
Find	ProcessAppMsg	
GetDocManager	PumpWaitingMessages	
GetMainWindow	QueryThrow	

TBitMap gdiobjec.h

TGdiObject

GetObject
IsGdiObject
IsOk
RefAdd
RefCount
RefDec
RefFind
RefInc
RefRemove

TBitMap

BitsPixel
GetBitmapBits
GetBitmapDimension
GetObject
HandleCreate
Height
Planes
SetBitmapBits
SetBitmapDimension
ToClipboard
Width

TBitMapGadget bitmapga.h

TGadget

CommandEnable	Invalidate	Removed
GetBorders	InvalidateRect	SetBorders
GetBorderStyle	LButtonDown	SetBorderStyle
GetBounds	LButtonUp	*SetBounds*
GetDesiredSize	MouseEnter	SetEnabled
GetEnabled	MouseLeave	SetMargins
GetId	MouseMove	SetShrinkWrap
GetInnerRect	NextGadget	SetSize
GetMargins	*Paint*	*SysColorChange*
GetOuterSizes	PaintBorder	
Inserted	PtIn	

TBitmapGadget

GetDesiredSize
Paint
SelectImage
SetBounds
SysColorChange

TButton button.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TControl

CompareItem	EvPaint	ODAFocus
DeleteItem	MeasureItem	ODASelect
DrawItem	ODADrawEntire	

TButton

BMSetStyle
EvGetDlgCode
GetClassName
SetupWindow

TButtonGadget buttonga.h

TGadget

CommandEnable	*Invalidate*	Removed
GetBorders	InvalidateRect	SetBorders
GetBorderStyle	*LButtonDown*	SetBorderStyle
GetBounds	*LButtonUp*	SetBounds
GetDesiredSize	MouseEnter	SetEnabled
GetEnabled	MouseLeave	SetMargins
GetId	MouseMove	SetShrinkWrap
GetInnerRect	NextGadget	SetSize
GetMargins	*Paint*	*SysColorChange*
GetOuterSizes	PaintBorder	
Inserted	PtIn	

TButtonGadget

Activate	LButtonDown
BeginPressed	LButtonUp
CancelPressed	Paint
CommandEnable	SetButtonState
GetButtonState	SetBounds
GetButtonType	SetNotchCorners
GetDesiredSize	SetShadowStyle
Invalidate	SysColorChange

TCheckbox checkbox.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TControl

CompareItem	EvPaint	ODAFocus
DeleteItem	MeasureItem	ODASelect
DrawItem	ODADrawEntire	

TButton

BMSetStyle
EvGetDlgCode
GetClassName
SetupWindow

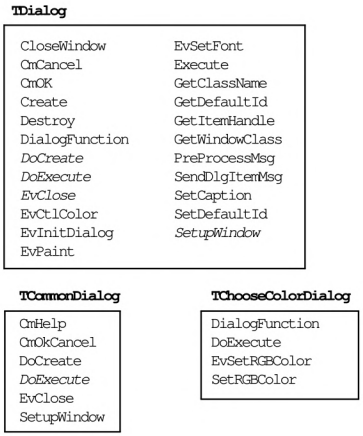
TCheckBox

BNClicked	SetCheck
Check	SetState
EvGetDlgCode	SetStyle
GetClassName	Toggle
GetCheck	Transfer
GetState	Uncheck

TChooseColorDialog chooseco.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	*GetWindowClass*	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	*SetupWindow*
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp


TDialog
CloseWindow
CmCancel
CmOK
Create
Destroy
DialogFunction
DoCreate
DoExecute
EvClose
EvCtlColor
EvInitDialog
EvPaint
EvSetFont
Execute
GetClassName
GetDefaultId
GetItemHandle
GetWindowClass
PreProcessMsg
SendDlgItemMsg
SetCaption
SetDefaultId
SetupWindow
TCommonDialog
CmHelp
CmOkCancel
DoCreate
DoExecute
EvClose
SetupWindow
TChooseColorDialog
DialogFunction
DoExecute
EvSetRGBColor
SetRGBColor

TChooseFontDialog choosefo.h

TWindow

AdjustWindowRect	GetSystemMenu	ScrollWindowEx
AdjustWindowRectEx	GetTopWindow	*SendDlgItemMessage*
BringWindowToTop	GetUpdateRect	SendMessage
CanClose	GetUpdateRgn	SendNotification
CheckDlgButton	*GetWindowClass*	SetActiveWindow
CheckRadioButton	GetWindowLong	SetBkgndColor
ChildBroadcastMessage	GetWindowPlacement	*SetCaption*
ChildWindowFromPoint	GetWindowPtr	SetCaretBlinkTime
ChildWithId	GetWindowRect	SetCaretPos
CleanupWindow	GetWindowTask	SetClassLong
ClearFlag	GetWindowText	SetClassWord
ClientToScreen	GetWindowTextLength	SetCursor
CloseWindow	GetWindowTextTitle	SetDlgItem
CmExist	GetWindowWord	SetDlgItemText
Create	HandleMessage	SetFlag
CreateCaret	HideCaret	SetModule
DefaultProcessing	HiliteMenuItem	SetMenu
DefWindowProc	HoldFocusHwnd	SetParent
Destroy	Invalidate	SetProp
DestroyCaret	InvalidateRect	SetScrollPos
DisableAutoCreate	InvalidateRgn	SetScrollRange
DisableTransfer	IsChild	SetSysModalWindow
DispatchScroll	IsDlgButtonChecked	SetTimer
GetChildren	IsFlagSet	SetTransferBuffer
GetClassLong	IsIconic	*SetupWindow*
GetClassName	IswindowVisible	SetWindowLong
GetClassWord	IsZoomed	SetWindowPlacement
GetClientRect	KillTimer	SetWindowPos
GetCursorPos	LoadAcceleratorTable	SetWindowText
GetDesktopWindow	LockWindowUpdate	SetWindowWord
GetDlgItem	MapWindowPoints	Show
GetDlgItemInt	MessageBox	ShowCaret
GetDlgItemText	MoveWindow	ShowOwnedPopups
GetFirstChild	Next	ShowScrollBar
GetId	NumChildren	ShowWindow
GetModule	OpenClipboard	SubclassWindowFunction
GetLastActivePopup	Paint	Transfer
GetLastChild	PostMessage	TransferData
GetMenu	PerformCreate	UnregisterHotKey
GetNextDlgGroupItem	*PreProcessMsg*	UpdateWindow
GetNextDlgTabItem	Previous	Validate
GetNextWindow	ReceiveMessage	ValidateRect
GetObjectPtr	RedrawWindow	ValidateRgn
GetParent	Register	WindowFromPoint
GetProp	RegisterHotKey	WindowProc
GetScrollPos	RemoveProp	WinHelp
GetScrollRange	ScreenToClient	
GetSysModalWindow	ScrollWindow	

TDialog

CloseWindow	EvSetFont
CmCancel	Execute
CmOK	GetClassName
Create	GetDefaultId
Destroy	GetItemHandle
DialogFunction	GetWindowClass
DoCreate	PreProcessMsg
DoExecute	SendDlgItemMsg
EvClose	SetCaption
EvCtlColor	SetDefaultId
EvInitDialog	*SetupWindow*
EvPaint	

TCommonDialog

CmHelp
CmOkCancel
DoCreate
DoExecute
EvClose
SetupWindow

TChooseFontDialog

CmFontApply
DialogFunction
DoExecute

TClipboardViewer clipview.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	*SetupWindow*
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TClipboardViewer

EvChangeCBChain
EvDestroy
EvDrawClipboard
SetupWindow

TComboBox combobox.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TControl

CompareItem	EvPaint	ODAFocus
DeleteItem	MeasureItem	ODASelect
DrawItem	ODADrawEntire	

TListBox

AddString	GetSel	*SetItemData*
ClearList	GetSelCount	*SetItemHeight*
DeleteString	*GetSelIndex*	SetSel
DirectoryList	GetSelIndexes	*SetSelIndex*
FindExactString	GetSelString	SetSelIndexes
FindString	GetSelStrings	SetSelItemRange
GetCaretIndex	*GetString*	*SetSelString*
GetClassName	*GetStringLen*	SetSelStrings
GetCount	GetTopIndex	SetTabStops
GetHorizontalExtent	*InsertString*	SetTopIndex
GetItemData	SetCaretIndex	*Transfer*
GetItemHeight	SetColumnWidth	
GetItemRect	SetHorizontalExtent	

TComboBox

AddString	GetExtendedUI	SetExtendedUI
Clear	GetItemData	SetItemData
ClearList	GetItemHeight	SetItemHeight
DeleteString	GetSelIndex	SetSelIndex
DirectoryList	GetString	SetSelString
FindString	GetStingLen	SetText
GetClassName	GetText	SetupWindow
GetCount	GetTextLen	ShowList
GetDroppedControlRect	HideList	Transfer
GetDroppedState	InsertString	
GetEditSel	SetEditSel	

TComboBoxData combobox.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TControl

CompareItem	EvPaint	ODAFocus
DeleteItem	MeasureItem	ODASelect
DrawItem	ODADrawEntire	

TListBox

AddString	GetSel	*SetItemData*
ClearList	GetSelCount	*SetItemHeight*
DeleteString	*GetSelIndex*	SetSel
DirectoryList	GetSelIndexes	*SetSelIndex*
FindExactString	GetSelString	SetSelIndexes
FindString	GetSelStrings	SetSelItemRange
GetCaretIndex	*GetString*	*SetSelString*
GetClassName	*GetStringLen*	SetSelStrings
GetCount	GetTopIndex	SetTabStops
GetHorizontalExtent	*InsertString*	SetTopIndex
GetItemData	SetCaretIndex	*Transfer*
GetItemHeight	SetColumnWidth	
GetItemRect	SetHorizontalExtent	

TComboBox

AddString	GetExtendedUI	SetExtendedUI
Clear	GetItemData	SetItemData
ClearList	GetItemHeight	SetItemHeight
DeleteString	GetSelIndex	SetSelIndex
DirectoryList	GetString	SetSelString
FindString	GetStingLen	SetText
GetClassName	GetText	SetupWindow
GetCount	GetTextLen	ShowList
GetDroppedControlRect	HideList	Transfer
GetDroppedState	InsertString	
GetEditSel	SetEditSel	

TComboBoxData

AddItemData
AddString
AddStringItem

TCommonDialog commdial.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	*SendDlgItemMessage*
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	*SetCaption*
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	*GetWindowClass*	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	*SetupWindow*
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	*PreProcessMsg*	WindowProc
GetLastActivePopup	Previous	WinHelp

TDialog

CloseWindow	EvSetFont
CmCancel	Execute
CmOK	GetClassName
Create	GetDefaultId
Destroy	GetItemHandle
DialogFunction	GetWindowClass
DoCreate	PreProcessMsg
DoExecute	SendDlgItemMsg
EvClose	SetCaption
EvCtlColor	SetDefaultId
EvInitDialog	*SetupWindow*
EvPaint	

TCommonDialog

CmHelp
CmOkCancel
DoCreate
DoExecute
EvClose
SetupWindow

TControl control.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TControl

CompareItem	EvPaint	ODAFocus
DeleteItem	MeasureItem	ODASelect
DrawItem	ODADrawEntire	

TControlBar controlb.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	*Paint*	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TGadgetWindow

Create	GetInnerRect
EvLButtonDown	GetMargins
EvLButtonUp	IdleActionInsert
EvMouseMove	LayoutSession
EvSize	LayoutUnitsToPixels
EvSysColorChange	NextGadget
FirstGadget	Paint
GadgetChangedSize	PaintGadgets
GadgetFromPoint	*PositionGadget*
GadgetReleaseCapture	Remove
GadgetSetCapture	SetDirection
GadgetWithId	SetHintCommand
GetDesiredSize	SetHintMode
GetDirection	SetMargins
GetFont	SetShrinkWrap
GetFontHeight	TileGadgets
GetHintMode	

TControlBar

PreProcessMsg
PositionGadget

TControlGadget controlg.h

TGadget

CommandEnable	*Invalidate*	*Removed*
GetBorders	*InvalidateRect*	SetBorders
GetBorderStyle	LButtonDown	SetBorderStyle
GetBounds	LButtonUp	*SetBounds*
GetDesiredSize	MouseEnter	SetEnabled
GetEnabled	MouseLeave	SetMargins
GetId	MouseMove	SetShrinkWrap
GetInnerRect	NextGadget	SetSize
GetMargins	Paint	SysColorChange
GetOuterSizes	PaintBorder	
Inserted	PtIn	

TControlGadget

GetDesiredSize
GetInnerRect
Inserted
Invalidate
InvalidateRect
Removed
SetBounds
Update

TDecoratedFrame decframe.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	*HoldFocusHwnd*	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	*SetupWindow*
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	*PreProcessMsg*	WindowProc
GetLastActivePopup	Previous	WinHelp

TFrameWindow

AssignMenu	EvQueryDragIcon	*PreProcessMsg*
EnableKBHandler	EvSetFocus	RestoreMenu
EvCommand	*EvSize*	SetClientWindow
EvCommandEnable	GetClientWindow	SetIcon
EvEraseBkgnd	GetMenuDescr	SetMenuDescr
EvInitMenuPopup	HoldFocusHwnd	*SetupWindow*
EvPaint	IdleAction	
EvParentNotify	MergeMenu	

TDecoratedFrame

EvCommand	EvSize
EvCommandEnable	Insert
EvEnterIdle	PreProcessMsg
EvMenuSelect	SetupWindow

TDecoratedMDIFrame decmdifr.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	*HoldFocusHwnd*	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	*SetupWindow*
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	*PreProcessMsg*	WindowProc
GetLastActivePopup	Previous	WinHelp

TFrameWindow

AssignMenu	EvQueryDragIcon	*PreProcessMsg*
EnableKBHandler	EvSetFocus	RestoreMenu
EvCommand	*EvSize*	SetClientWindow
EvCommandEnable	GetClientWindow	SetIcon
EvEraseBkgnd	GetMenuDescr	SetMenuDescr
EvInitMenuPopup	HoldFocusHwnd	*SetupWindow*
EvPaint	IdleAction	
EvParentNotify	MergeMenu	

TDecoratedFrame

EvCommand	EvSize
EvCommandEnable	Insert
EvEnterIdle	PreProcessMsg
EvMenuSelect	SetupWindow

TMDIFrame

DefWindowProc
GetClientWindow
SetMenu

TDecoratedMDIFrame

DefWindowProc

TDialog dialog.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	*GetWindowClass*	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	*HoldFocusHwnd*	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	*SetupWindow*
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	*Paint*	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TDialog

CloseWindow	EvSetFont
CmCancel	Execute
CmOK	GetClassName
Create	GetDefaultId
Destroy	GetItemHandle
DialogFunction	GetWindowClass
DoCreate	PreProcessMsg
DoExecute	SendDlgItemMsg
EvClose	SetCaption
EvCtlColor	SetDefaultId
EvInitDialog	SetupWindow
EvPaint	

TDocManager docmanag.h

TDocManager

CmFileClose	CreateAnyView	PostDocError
CmFileNew	DeleteTemplate	PostEvent
CmFileOpen	EvCanClose	RefTemplate
CmFileRevert	EvPreProcessMenu	SelectAnySave
CmFileSave	FlushDoc	SelectDocPath
CmFileSaveAs	GetApplication	SelectDocType
CmViewCreate	GetCurrentDoc	SelectViewType
CreateAnyDoc	MatchTemplate	UnRefTemplate

TDocTemplate docmanag.h

TDocTemplate

ClearFlag	GetFileFilter	IsVisible
CreateDoc	GetFlags	SelectSave
CreateView	GetViewName	SetDefaultExt
GetDefaultExt	InitDoc	SetDirectory
GetDescription	InitView	SetDocManager
GetDirectory	IsFlagSet	SetFileFilter
GetDocManager	IsMyKindOfDoc	SetFlag

TDocument docview.h

TDocument

AttachStream	GetTitle	PropertyFlags
CanClose	HasFocus	PropertyName
Close	InStream	QueryViews
Commit	IsDirty	Revert
DetachStream	IsOpen	RootDocument
FindProperty	NextStream	SetDocManager
GetDocManager	NextView	SetDocPath
GetDocPath	NotifyViews	SetOpenMode
GetOpenMode	Open	SetProperty
GetParentDoc	OutStream	SetTemplate
GetProperty	PostError	SetTitle
GetTemplate	PropertyCount	

TEdit edit.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TControl

CompareItem	EvPaint	ODAFocus
DeleteItem	MeasureItem	ODASelect
DrawItem	ODADrawEntire	

TStatic

Clear
GetClassName
GetText
GetTextLen
SetText
Transfer

TEdit

CanClose	EvGetDlgCode	Insert
CanUndo	EvKeyDown	IsModified
ClearModify	EvKillFocus	LockBuffer
CmEditClear	FormatLines	Paste
CmEditCopy	GetClassName	Scroll
CmEditCut	GetFirstVisibleLine	Search
CmEditDelete	GetHandle	SetHandle
CmEditPaste	GetLine	SetPasswordChar
CmEditUndo	GetLineFromPos	SetReadOnly
Copy	GetLineIndex	SetRect
Cut	GetLineLength	SetRectNP
DeleteLine	GetNumLines	SetSelection
DeleteSelection	GetPasswordChar	SetTabStops
DeleteSubText	GetRect	SetupWindow
EmptyUndoBuffer	GetSelection	SetWordBreakProc
ENErrSpace	GetSubText	Undo
EvChar	GetWordBreakProc	UnlockBuffer

TEditFile editfile.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TControl

CompareItem	EvPaint	ODAFocus
DeleteItem	MeasureItem	ODASelect
DrawItem	ODADrawEntire	

TStatic

Clear
GetClassName
GetText
GetTextLen
SetText
Transfer

TEdit

CanClose	EvGetDlgCode	Insert
CanUndo	EvKeyDown	IsModified
ClearModify	EvKillFocus	LockBuffer
CmEditClear	FormatLines	Paste
CmEditCopy	GetClassName	Scroll
CmEditCut	GetFirstVisibleLine	Search
CmEditDelete	GetHandle	SetHandle
CmEditPaste	GetLine	SetPasswordChar
CmEditUndo	GetLineFromPos	SetReadOnly
Copy	GetLineIndex	SetRect
Cut	GetLineLength	SetRectNP
DeleteLine	GetNumLines	SetSelection
DeleteSelection	GetPasswordChar	SetTabStops
DeleteSubText	GetRect	*SetupWindow*
EmptyUndoBuffer	GetSelection	SetWordBreakProc
ENErrSpace	GetSubText	Undo
EvChar	GetWordBreakProc	UnlockBuffer

TEditSearch

CmEditFind
CmEditFindNext
CmEditReplace
DoSearch
EvFindMsg
SetupWindow

TEditFile

CanClear	CmFileSaveAs	Save
CanClose	NewFile	SaveAs
CmFileNew	Open	SetFileName
CmFileOpen	Read	SetupWindow
CmFileSave	ReplaceWith	Write

TEditSearch editsear.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TControl

CompareItem	EvPaint	ODAFocus
DeleteItem	MeasureItem	ODASelect
DrawItem	ODADrawEntire	

TStatic

Clear
GetClassName
GetText
GetTextLen
SetText
Transfer

TEdit

CanClose	EvGetDlgCode	Insert
CanUndo	EvKeyDown	IsModified
ClearModify	EvKillFocus	LockBuffer
CmEditClear	FormatLines	Paste
CmEditCopy	GetClassName	Scroll
CmEditCut	GetFirstVisibleLine	Search
CmEditDelete	GetHandle	SetHandle
CmEditPaste	GetLine	SetPasswordChar
CmEditUndo	GetLineFromPos	SetReadOnly
Copy	GetLineIndex	SetRect
Cut	GetLineLength	SetRectNP
DeleteLine	GetNumLines	SetSelection
DeleteSelection	GetPasswordChar	SetTabStops
DeleteSubText	GetRect	*SetupWindow*
EmptyUndoBuffer	GetSelection	SetWordBreakProc
ENErrSpace	GetSubText	Undo
EvChar	GetWordBreakProc	UnlockBuffer

TEditSearch

CmEditFind
CmEditFindNext
CmEditReplace
DoSearch
EvFindMsg
SetupWindow

TEditView editview.h

TView

FindProperty	IsOK
GetDocument	NotOK
GetNextViewId	PropertyCount
GetProperty	PropertyFlags
GetViewId	PropertyName
GetViewMenu	*SetDocTitle*
GetViewName	SetProperty
GetWindow	SetViewMenu

TEditSearch

CmEditFind
CmEditFindNext
CmEditReplace
DoSearch
EvFindMsg
SetupWindow

TEditView

CanClose	LoadData	VnDocClosed
Create	PerformCreate	VnIsDirty
EvNCDestroy	SetDocTitle	VnIsWindow
GetViewName	StaticName	VnRevert
GetWindow	VnCommit	

TFileDocument filedoc.h

TDocument

AttachStream	GetTitle	*PropertyFlags*
CanClose	HasFocus	*PropertyName*
Close	*InStream*	QueryViews
Commit	IsDirty	*Revert*
DetachStream	*IsOpen*	RootDocument
FindProperty	NextStream	SetDocManager
GetDocManager	NextView	SetDocPath
GetDocPath	NotifyViews	SetOpenMode
GetOpenMode	*Open*	*SetProperty*
GetParentDoc	*OutStream*	SetTemplate
GetProperty	PostError	SetTitle
GetTemplate	PropertyCount	

TFileDocument

Close	InStream	PropertyFlags
CloseThisFile	IsOpen	PropertyName
Commit	Open	Revert
FindProperty	OpenThisFile	SetProperty
GetProperty	OutStream	

TFileOpenDialog opensave.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	*GetWindowClass*	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	*HoldFocusHwnd*	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	*SetupWindow*
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TDialog

CloseWindow	EvSetFont
CmCancel	Execute
CmOK	GetClassName
Create	GetDefaultId
Destroy	GetItemHandle
DialogFunction	GetWindowClass
DoCreate	PreProcessMsg
DoExecute	SendDlgItemMsg
EvClose	SetCaption
EvCtlColor	SetDefaultId
EvInitDialog	*SetupWindow*
EvPaint	

TCommonDialog

CmHelp
CmOkCancel
DoCreate
DoExecute
EvClose
SetupWindow

TOpenSaveDialog

CmLbSelChanged
CmOk
DialogFunction
DoExecute
GetFileTitle
GetFileTitleLen
Init
ShareViolation

TFileOpenDialog

DoExecute

TFileSaveDialog opensave.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	*GetWindowClass*	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	*HoldFocusHwnd*	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	*SetupWindow*
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TDialog

CloseWindow	EvSetFont
CmCancel	Execute
CmOK	GetClassName
Create	GetDefaultId
Destroy	GetItemHandle
DialogFunction	GetWindowClass
DoCreate	PreProcessMsg
DoExecute	SendDlgItemMsg
EvClose	SetCaption
EvCtlColor	SetDefaultId
EvInitDialog	*SetupWindow*
EvPaint	

TCommonDialog

CmHelp
CmOkCancel
DoCreate
DoExecute
EvClose
SetupWindow

TOpenSaveDialog

CmLbSelChanged
CmOk
DialogFunction
DoExecute
GetFileTitle
GetFileTitleLen
Init
ShareViolation

TFileSaveDialog

DoExecute

TFindDialog findrepl.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	*GetWindowClass*	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	*SetupWindow*
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TDialog

CloseWindow	EvSetFont
CmCancel	Execute
CmOK	GetClassName
Create	GetDefaultId
Destroy	GetItemHandle
DialogFunction	GetWindowClass
DoCreate	PreProcessMsg
DoExecute	SendDlgItemMsg
EvClose	SetCaption
EvCtlColor	SetDefaultId
EvInitDialog	*SetupWindow*
EvPaint	

TCommonDialog

CmHelp
CmOkCancel
DoCreate
DoExecute
EvClose
SetupWindow

TFindReplaceDialog

CmFindNext
CmReplace
CmReplaceAll
CmCancel
DoCreate
EvNCDestroy
IsValid
IsValidInput

TFindDialog

DoCreate

TFloatingFrame floatfra.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	*HoldFocusHwnd*	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	*SetupWindow*
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	*PreProcessMsg*	WindowProc
GetLastActivePopup	Previous	WinHelp

TTinyCaption

DoCommand	EnableTinyCaption	GetCaptionRect
DoLButtonUp	EvCommand	GetMinBoxRect
DoMouseMove	EvLButtonUp	GetMaxBoxRect
DoNCActivate	EvMouseMove	GetSysBoxRect
DoNCCalcSize	EvNCActivate	PaintButton
DoNCHitTest	EvNCCalcSize	PaintCaption
DoNCLButtonDown	EvNCHitTest	PaintCloseBox
DoNCPaint	EvNCLButtonDown	PaintMaxBox
DoSysCommand	EvNCPaint	PaintMinBox
DoSysMenu	EvSysCommand	PaintSysBox

TFrameWindow

AssignMenu	EvQueryDragIcon	PreProcessMsg
EnableKBHandler	EvSetFocus	RestoreMenu
EvCommand	EvSize	SetClientWindow
EvCommandEnable	GetClientWindow	SetIcon
EvEraseBkgnd	GetMenuDescr	SetMenuDescr
EvInitMenuPopup	HoldFocusHwnd	SetupWindow
EvPaint	IdleAction	
EvParentNotify	MergeMenu	

TFloatingFrame

SetMargins

TGadgetWindow gadgetwi.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	*Paint*	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TGadgetWindow

Create	GetInnerRect
EvLButtonDown	GetMargins
EvLButtonUp	IdleActionInsert
EvMouseMove	Insert
EvSize	LayoutSession
EvSysColorChange	LayoutUnitsToPixels
FirstGadget	NextGadget
GadgetChangedSize	Paint
GadgetFromPoint	PaintGadgets
GadgetReleaseCapture	PositionGadget
GadgetSetCapture	Remove
GadgetWithId	SetDirection
GetDesiredSize	SetHintCommand
GetDirection	SetHintMode
GetFont	SetMargins
GetFontHeight	SetShrinkWrap
GetHintMode	TileGadgets

TGauge gauge.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TControl

CompareItem	EvPaint	ODAFocus
DeleteItem	MeasureItem	ODASelect
DrawItem	ODADrawEntire	

TGauge

EvEraseBkgnd
Paint
PaintBorder

TLayoutWindow layoutwi.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TLayoutWindow

EvSize
GetChildLayoutMetrics
Layout
RemoveChildLayoutMetrics
SetChildLayoutMetrics

TListBox listbox.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TControl

CompareItem	EvPaint	ODAFocus
DeleteItem	MeasureItem	ODASelect
DrawItem	ODADrawEntire	

TListBox

AddString	GetSel	SetItemData
ClearList	GetSelCount	SetItemHeight
DeleteString	GetSelIndex	SetSel
DirectoryList	GetSelIndexes	SetSelIndex
FindExactString	GetSelString	SetSelIndexes
FindString	GetSelStrings	SetSelItemRange
GetCaretIndex	GetString	SetSelString
GetClassName	GetStringLen	SetSelStrings
GetCount	GetTopIndex	SetTabStops
GetHorizontalExtent	InsertString	SetTopIndex
GetItemData	SetCaretIndex	Transfer
GetItemHeight	SetColumnWidth	
GetItemRect	SetHorizontalExtent	

TListView listview.h

TView

FindProperty	IsOK
GetDocument	NotOK
GetNextViewId	PropertyCount
GetProperty	PropertyFlags
GetViewId	PropertyName
GetViewMenu	*SetDocTitle*
GetViewName	SetProperty
GetWindow	SetViewMenu

TListBox

AddString	GetSel	SetItemData
ClearList	GetSelCount	SetItemHeight
DeleteString	GetSelIndex	SetSel
DirectoryList	GetSelIndexes	SetSelIndex
FindExactString	GetSelString	SetSelIndexes
FindString	GetSelStrings	SetSelItemRange
GetCaretIndex	GetString	SetSelString
GetClassName	GetStringLen	SetSelStrings
GetCount	GetTopIndex	SetTabStops
GetHorizontalExtent	InsertString	SetTopIndex
GetItemData	SetCaretIndex	Transfer
GetItemHeight	SetColumnWidth	
GetItemRect	SetHorizontalExtent	

TListView

CanClose	CmEditUndo	SetExtent
CmEditAdd	CmSelChange	StaticName
CmEditCopy	Create	VnCommit
CmEditClear	EvGetDlgCode	VnDocClosed
CmEditCut	GetViewName	VnIsDirty
CmEditDelete	GetWindow	VnIsWindow
CmEditItem	LoadData	VnRevert
CmEditPaste	SetDocTitle	

TMDIClient mdi.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	*PreProcessMsg*	WindowProc
GetLastActivePopup	Previous	WinHelp

TMDIClient

ArrangeIcons	CmCloseChildren	EvMDICreate
CascadeChildren	CmCreateChild	GetActiveMDIChild
CloseChildren	CmTileChildren	GetClassName
CmArrangeIcons	CmTileChildrenHoriz	InitChild
CmCascadeChildren	Create	PreProcessMsg
CmChildActionEnable	CreateChild	TileChildren

TMDIFrame mdi.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	*HoldFocusHwnd*	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	*SetupWindow*
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	*PreProcessMsg*	WindowProc
GetLastActivePopup	Previous	WinHelp

TTinyCaption

DoCommand	EnableTinyCaption	GetCaptionRect
DoLButtonUp	EvCommand	GetMinBoxRect
DoMouseMove	EvLButtonUp	GetMaxBoxRect
DoNCActivate	EvMouseMove	GetSysBoxRect
DoNCCalcSize	EvNCActivate	PaintButton
DoNCHitTest	EvNCCalcSize	PaintCaption
DoNCLButtonDown	EvNCHitTest	PaintCloseBox
DoNCPaint	EvNCLButtonDown	PaintMaxBox
DoSysCommand	EvNCPaint	PaintMinBox
DoSysMenu	EvSysCommand	PaintSysBox

TFrameWindow

AssignMenu	EvQueryDragIcon	PreProcessMsg
EnableKBHandler	EvSetFocus	RestoreMenu
EvCommand	EvSize	SetClientWindow
EvCommandEnable	*GetClientWindow*	SetIcon
EvEraseBkgnd	GetMenuDescr	SetMenuDescr
EvInitMenuPopup	HoldFocusHwnd	SetupWindow
EvPaint	IdleAction	
EvParentNotify	MergeMenu	

TMDIFrame

DefWindowProc
GetClientWindow
SetMenu

TRadioButton radiobut.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TControl

CompareItem	EvPaint	ODAFocus
DeleteItem	MeasureItem	ODASelect
DrawItem	ODADrawEntire	

TButton

BMSetStyle
EvGetDlgCode
GetClassName
SetupWindow

TCheckBox

BNClicked	SetCheck
Check	SetState
EvGetDlgCode	SetStyle
GetClassName	Toggle
GetCheck	Transfer
GetState	Uncheck

TRadioButton

BNClicked

TSlider slider.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TControl

CompareItem	EvPaint	ODAFocus
DeleteItem	MeasureItem	ODASelect
DrawItem	ODADrawEntire	

TScrollBar

DeltaPos	SBPageUp
GetClassName	SBThumbPosition
GetPosition	SBThumbTrack
GetRange	SBTop
SBBottom	SetPosition
SBLineDown	SetRange
SBLineUp	*SetupWindow*
SBPageDown	Transfer

TSlider

EvEraseBkgnd	EvSize	PosToPoint
EvGetDlgCode	GetBkColor	SetPosition
EvKeyDown	GetPosition	SetRange
EvKillFocus	GetRange	SetRuler
EvLButtonDblClk	HitTest	SetupThumbRgn
EvLButtonDown	NotifyParent	SetupWindow
EvLButtonUp	PaintRuler	SlideThumb
EvMouseMove	PaintSlot	SnapPos
EvPaint	PaintThumb	
EvSetFocus	PointToPos	

TStatusBar statusba.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	*Paint*	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TGadgetWindow

Create	*GetInnerRect*
EvLButtonDown	GetMargins
EvLButtonUp	IdleActionInsert
EvMouseMove	Insert
EvSize	LayoutSession
EvsysColorChange	LayoutUnitsToPixels
FirstGadget	NextGadget
GadgetChangedSize	Paint
GadgetFromPoint	*PaintGadgets*
GadgetReleaseCapture	PositionGadget
GadgetSetCapture	Remove
GadgetWithId	SetDirection
GetDesiredSize	SetHintCommand
GetDirection	SetHintMode
GetFont	SetMargins
GetFontHeight	SetShrinkWrap
GetHintMode	TileGadgets

TMessageBar

GetDesiredSize
GetInnerRect
PaintGadgets
SetHintText
SetText

TStatusBar

Insert
PositionGadget
SetModeIndicator
ToggleModeIndicator
SetSpacing

TTextGadget textgadg.h

TGadget

CommandEnable	*Invalidate*	Removed
GetBorders	InvalidateRect	SetBorders
GetBorderStyle	LButtonDown	SetBorderStyle
GetBounds	LButtonUp	SetBounds
GetDesiredSize	MouseEnter	SetEnabled
GetEnabled	MouseLeave	SetMargins
GetId	MouseMove	SetShrinkWrap
GetInnerRect	NextGadget	SetSize
GetMargins	*Paint*	SysColorChange
GetOuterSizes	PaintBorder	
Inserted	PtIn	

TTextGadget

GetDesiredSize
GetText
Invalidate
Paint
SetText

TToolBox toolbox.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	*Paint*	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TGadgetWindow

Create	GetInnerRect
EvLButtonDown	GetMargins
EvLButtonUp	IdleActionInsert
EvMouseMove	*Insert*
EvSize	LayoutSession
EvsysColorChange	LayoutUnitsToPixels
FirstGadget	NextGadget
GadgetChangedSize	Paint
GadgetFromPoint	PaintGadgets
GadgetReleaseCapture	PositionGadget
GadgetSetCapture	Remove
GadgetWithId	SetDirection
GetDesiredSize	SetHintCommand
GetDirection	SetHintMode
GetFont	SetMargins
GetFontHeight	SetShrinkWrap
GetHintMode	*TileGadgets*

TToolBox

GetDesiredSize
Insert
TileGadgets

TWindowView docview.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

TView

FindProperty	IsOK
GetDocument	NotOK
GetNextViewId	PropertyCount
GetProperty	PropertyFlags
GetViewId	PropertyName
GetViewMenu	*SetDocTitle*
GetViewName	SetProperty
GetWindow	SetViewMenu

TWindowView

CanClose
GetViewName
GetWindow
SetDocTitle
StaticName

TWindow window.h

TWindow

AdjustWindowRect	GetLastChild	ReceiveMessage
AdjustWindowRectEx	GetMenu	RedrawWindow
BringWindowToTop	GetModule	Register
CanClose	GetNextDlgGroupIt	RegisterHotKey
CheckDlgButton	GetNextDlgTabItem	RemoveProp
CheckRadioButton	GetNextWindow	ScreenToClient
ChildBroadcastMessage	GetObjectPtr	ScrollWindow
ChildWindowFromPoint	GetParent	ScrollWindowEx
ChildWithId	GetProp	SendDlgItemMessage
CleanupWindow	GetScrollPos	SendMessage
ClearFlag	GetScrollRange	SendNotification
ClientToScreen	GetSysModalWindow	SetActiveWindow
CloseWindow	GetSystemMenu	SetBkgndColor
CmExist	GetTopWindow	SetCaption
Create	GetUpdateRect	SetCaretBlinkTime
CreateCaret	GetUpdateRgn	SetCaretPos
DefaultProcessing	GetWindowClass	SetClassLong
DefWindowProc	GetWindowLong	SetClassWord
Destroy	GetWindowPlacement	SetCursor
DestroyCaret	GetWindowPtr	SetDlgItem
DisableAutoCreate	GetWindowRect	SetDlgItemText
DisableTransfer	GetWindowTask	SetFlag
DispatchScroll	GetWindowText	SetModule
DragAcceptFiles	GetWindowTextLength	SetMenu
DrawMenuBar	GetWindowTextTitle	SetParent
EnableAutoCreate	GetWindowWord	SetProp
EnableTransfer	HandleMessage	SetScrollPos
EnumProps	HideCaret	SetScrollRange
EvChildInvalid	HiliteMenuItem	SetSysModalWindow
EvCommand	HoldFocusHwnd	SetTimer
EvCommandEnable	Invalidate	SetTransferBuffer
EvSysCommand	InvalidateRect	SetupWindow
FirstThat	InvalidateRgn	SetWindowLong
FlashWindow	IsChild	SetWindowPlacement
ForEach	IsDlgButtonChecked	SetWindowPos
ForwardMessage	IsFlagSet	SetWindowText
GetActiveWindow	IsIconic	SetWindowWord
GetApplication	IsZoomed	Show
GetCaretBlinkTime	IsWindowVisible	ShowCaret
GetCaretPos	KillTimer	ShowOwnedPopups
GetChildren	LoadAcceleratorTable	ShowScrollBar
GetClassLong	LockWindowUpdate	ShowWindow
GetClassName	MapWindowPoints	SubclassWindowFunction
GetClassWord	MessageBox	Transfer
GetClientRect	MoveWindow	TransferData
GetCursorPos	Next	UnregisterHotKey
GetDesktopWindow	NumChildren	UpdateWindow
GetDlgItem	OpenClipboard	Validate
GetDlgItemInt	Paint	ValidateRect
GetDlgItemText	PostMessage	ValidateRgn
GetFirstChild	PerformCreate	WindowFromPoint
GetId	PreProcessMsg	WindowProc
GetLastActivePopup	Previous	WinHelp

Index

!

3-D Unterstützung *213, 215*
<
    TBitmap bezogener Operator *101*
    TClipboard bezogener Operator *101*
    TClipboard-bezogener Operator *223*
    TDib-bezogener Operator *223*
==
    TCursor Operator *161*
__BUILDOWLDLL Makro *79*
_AUTOCLASS Makro *618*
_OWLCLASS Makro *79*
_OWLDATA Makro *80*
_OWLDLL Makro *80*
_OWLFAR Makro *81*
_OWLFARVTABLE Makro *81*
_OWLFASTTHIS Makro *81*
_OWLFUNC Makro *81*
~TDocManager
    TDocManager Klassendestruktor *236*

A

ABC
    Struktur *829*
AbortDoc
    TPrintDC Elementfunktion *424*
Above
    TEdgeConstraint Elementfunktion *243*
Absolute
    TEdgeConstraint Elementfunktion *243*
    TEdgeOrSizeContraint Elementfunktion *244*
AccelTable
    TWindowAttr-Struktur *529*
AccessResource
    TModule Elementfunktion *368*
Activate
    TButtonGadget Elementfunktion *112*
Activate Elementfunktion
    TOcPart *730*
ActivatePart Elementfunktion
    TOcView *756*
ActivePart Datenelement
    TOcView *762*
Add
    TAppDictionary Elementfunktion *96*
    TSortedStringArray Elementfunktion *482*
Add Elementfunktion
    TOcNameList *727*
    TOcPartCollection *738*
AddItem
    TVbxControl Elementfunktion *515*
AddItemData
    TComboBox Elementfunktion *144*
AddString
    TComboBox Elementfunktion *140, 144*
    TListBox Elementfunktion *324*
    TListBoxData Elementfunktion *330*
AddStringItem
    TComboBox Elementfunktion *144*
    TListBoxData Elementfunktion *331*
AddUserFormatName
    TOleFrame Elementfunktion, öffentliche *381*
AddUserFormatName Elementfunktion
    TOcApp *696*
AdjustWindowRect
    TWindow Elementfunktion *533*
AdjustWindowRectEx
    TWindow Elementfunktion *533*
Aggregate Elementfunktion
    TUnknown *773*
Aggregation
    COM-Objekte *634, 638*
Aktionsschalter *56, 106*
    voreingestellte *106*
Aktivieren der 3-D Unterstützung *215*
Aktualisierungsbereich
    Fenster *548*
Alarmton *253*
Align
    TTextGadget Elementfunktion *498*
Alloc Elementfunktion
    TOleAllocator *766*
AllocResource
    TModule Elementfunktion *368*

AngleArc
TDC Elementfunktion *162*
AnimatePalette
TPalette Elementfunktion *415*
Animation *115*
Anordnungsrichtung *506*
AntialiasEdges
TButtonGadget Datenelement *111*
Antwortfunktionen *583, 585, 587, 589*
Antworttabelle
TFloatingFrame Klasse *276*
TOleView *391*
Antworttabellen
Definieren *60, 63*
Deklarieren *59*
TButton Klasse *108*
TCheckBox Klasse *121*
TChooseColorDialog Klasse *123*
TChooseFontDialog Klasse *125*
TClipboardViewer Klasse *134*
TCommonDialog Klasse *153*
TControl Klasse *159*
TControlGadget *156*
TDecoratedFrame Klasse *211*
TDecoratedMDIFrame Klasse *212*
TDialog Klasse *218*
TDocManager Klasse *242*
TEdit Klasse *254*
TEditFile Klasse *257*
TEditSearch Klasse *259*
TFrameWindow Klasse *284*
TGadgetWindow Klasse *299*
TGauge Klasse *302*
TLayoutWindow Klasse *323*
TListView Klasse *336*
TMDIChild Klasse *341*
TMDIClient Klasse *345*
TOleFrame *385*
TOleMDIFrame *388*
TOleWindow *407*
TRadioButton Klasse *454*
TScrollBar Klasse *467*
TStatic Klasse *486*
TTinyCaption Klasse *505*
TVbXControl *520*
TVbxEventHandler Klasse *525*
TWindow Klasse *573*
Antworttabellen und
Dokumentverwaltung *235*
Antworttabellen-Makros
für VBX-Ereignisse *522*
Antworttabelleneintäge *460*
Antworttabelleneinträge
TWindowView *575*
Antworttabellenmakro
und Funktionskopf-Template *591*
Anwendungen
Ereignisbotschaften *650*
Erleichtern *156*
Anzeige
aktueller Status *489*
AppDesc Datenelement
TRegistrar *772*
AppendMenu
TMenu Elementfunktion *350*
Arc
TDC Elementfunktion *163*
Area
TRect Elementfunktion *809*
ArgCount Datenelement
TAutoStack *683*
ArgSymbolCount Datenelement
TAutoStack *683*
Argumente
Automatisierungs-Server *657*
ArrangeIcons
TMDIClient Elementfunktion *342*
Arrays
Bitmap-Bilder *102*
Cels *115*
ArraySize
TSortedStringArray Elementfunktion *482*
Aspect Datenelement
TOcViewPaint *764*
aspectall Registrierschlüssel *614*
aspectcontent Registrierschlüssel *614*
aspectdocprint Registrierschlüssel *615*
aspecticon Registrierschlüssel *615*
aspectthumbnail Registrierschlüssel *616*
Aspekte
Präsentations- *652, 655*
AsPossible
TGadgetWindow Datenelement *297*
AssignMenu
TFrameWindow Elementfunktion *279*
asxxxx-Konstanten *627*
AtMouse
TGadgetWindow Datenelement *296*
AttachStream
TDocument Datenelement *234*

AttachTemplate
TDocManager Elementfunktion *237*
Attr
TDialog Datenelement *213*
TWindow Datenelement *531*
Attr Datenelement
TAutoCommand *665*
Attribute
Dokumenteigenschaften *79*
Editierfelder *246, 250*
Suchen-und-Ersetzen *271, 272*
Titelleisten *499*
Attributsliste
Handles ermitteln *547*
atxxxx-Konstanten *619*
Aufschneiden *115*
Aufzählung
TWidthHeight *528*
Aufzählungen
TRelationshipUnits *459*
Aufzählungstyp
AutoDataType *619*
TOcAppMode *705*
TOcAspect *704*
TOcDialogHelp *706*
TOcDropAction *713*
TOcInitHow *718*
TOcInitWhere *722*
TOcInvalidate *722*
TOcPartName *741*
TOcScrollDir *751*
Auslösen von Exception-Objekten *93*
Auswahl
Farben *121*
Schriften *124*
Text *249, 251*
AUTOARGS Makros *616*
AUTOCALL_xxxx Makros *617*
Autocreation
Oberflächenobjekte *538*
AUTODATA Makros *618*
AutoDataType Aufzählungstyp *619*
AUTODETACH Makro *620*
AUTOENUM Makros *620*
AUTOFLAG Makro *621*
AUTOFUNC Makros *621, 622*
AUTOINVOKE Makro *623*
AUTOITERATOR Makro *623*
automatisierte Methoden
Rückgabewerte *660*
Automatisierung
Datentypen *603*
Klassenmodifizierer *618*
ObjectComponents-Klassen *602*
Automatisierung von Objeken
Aktionsarten registrieren *654*
Automatisierung von Objekte
Aktionsarten registrieren *656*
Automatisierungkontroller *624*
Automatisierungs-Server
Anwendungen zugänglich machen *639*
Argumente *657*
Argumente Registrieren *631*
Argumente verifizieren *630*
Aufzählungstypen *620*
automatisierbare Elemente definieren *633, 635, 638, 639*
Bitflags *621*
Debuggen *633*
Deklarationsmakros *604, 605*
Elemente zugänglich machen *642*
Hook-Prozeduren *606, 623, 625, 626, 627, 629, 630*
Iteration *623, 641*
Kombinieren nicht verwandter Klassen *640*
Objekte entfernen *620*
Objekte zugänglich machen *629*
Zugriff auf Datenelemente *618*
Zugriff auf Eigenschaften *625*
Zugriff auf Elementfunktionen *621, 622, 628*
Zugriff auf Objekteigenschaften *643, 644*
Automatisierungskontroller *616, 617*
Platzhaltermakros *607*
AutoMode
TScroller Datenelement *468*
AUTONAMES Makro *624*
AUTONOHOOK Makro *625*
AutoOrg
TScroller Datenelement *468*
AUTOPROP Makro *625*
AUTORECORD Makro *626*
AUTOREPORT Makro *627*
AutoScroll
TScroller Elementfunktion *469*
AUTOSTAT Makro *628*
AutoSymFlag Aufzählungstyp *627*
AUTOTHIS Makro *629*
AUTOUNDO Makro *629*
AUTOVALIDATE Makro *630*

aveParts Elementfunktion
Siehe TOcDocument
aveToFile Elementfunktion
Siehe TOcDocument

B

BandInfo
TPrintDC Elementfunktion *425*
BANDINFOSTRUCT
Struktur *98*
Banding
TPrintout Datenelement *445*
BarColor
TGauge Datenelement *301*
Basisklasse
TAutoObjectDelete *676*
TOcApp *695*
TOcPart *728*
TOcRegistrar *741*
TOcRemView *744*
TOcView *755*
TXAuto *776*
TXObjComp *777*
TXOle *778*
TXRegistry *780*
Bedingungen
Kante *242*
Layout *317, 318, 319, 320, 321, 322*
BEGIN_REGISTRATION_makro *789*
BeginDocument
TPrintout Elementfunktion *443*
BeginModal
TApplication Elementfunktion *85*
BeginPressed
TButtonGadget Elementfunktion *112*
BeginPrinting
TPrintout Elementfunktion *443*
BeginView
TScroller Elementfunktion *469*
Behandlung des Eingabefokus *278*
Behandlung von Dokument-Views *227*
Beispiele
TLayoutWindow Klasse *320*
Beispielklasseneintrag *53, 54*
Beispielprogramme
Gleitregler *472*
VALIDATE.CPP *511*
Below
TEdgeConstraint Elementfunktion *243*
Benachrichtigungs-Codes *253*
Benannte Streams und TFileDocument *267*
Benutzereingabe
Dateneinträge prüfen *511*
numerische Werte prüfen *454*
Überprüfen *269*
Überprüfung *495*
Benutzereingabe prüfen
Eingabefelder *495*
Benutzeroberfläche
Bitmaps *115*
Bereichsprüfungsobjekte *454*
Bereichswerte
Konvertieren zu Bildlaufwerten *76*
BF_XXXX Schalter-Flag-Konstanten *56*
BF_XXXX Schalterbotschaftskonstanten *56*
Bibliotheken
DLL *44*
ObjectComponents *599*
Übersicht *44*
Versionsnummern, Zurückgeben *82*
Bildlaufleiste *463*
Warnung *463*
Bildlaufleisten *468*
Bereich *463*
Gleitregler vs. *472, 473, 474, 475, 476, 477, 478*
horizontale *463, 464, 465, 466*
Konstanten *82*
Position *463, 470*
Transfer *466*
TScrollBarData *467*
vertikale *463, 464, 465, 466*
Zeigerposition *467*
Bildlaufwerte
Konvertieren zu Bereichswerten *76*
Bind Elementfunktion
TAutoProxy *678*
Bit-Zahl
NBits *77*
BitBlt
TDC Elementfunktion *163*
Bitmap
Struktur *830*
TCelArray Datenelement *117*
Bitmap-Bilder
Anzeigen *102*
BITMAPCOREHEADER
Struktur *831*

BITMAPCOREINF
Struktur *831*
BITMAPINFO
Struktur *833*
BITMAPINFOHEADER
Struktur *835*
BitmapOrigin
TButtonGadget Datenelement *111*
Bitmaps
Benutzeroberfläche *115*
Löschen *116*
Referenzieren *116*
Bits
TDib Datenelement *225*
Bits löschen *103*
Bits setzen *103*
BitsPixel
TBitmap Elementfunktion *100*
Bkcolor
TSlider Elementfunktion *477*
BkgndBrush
TGadgetWindow Datenelement *296*
BkgndColor
TWindow Datenelement *568*
Black
TColor Datenelement *135*
Blue
TColor Elementfunktion *137*
BMSetStyle
TButton Elementfunktion *107*
BNClicked
TCheckBox Elementfunktion *120*
BOleComponentCreate Elementfunktion
TOcRegistrar *742*
Border
TTinyCaption Datenelement *500*
Borders
TGadget Datenelement *289*
BorderStyle
TGadget Datenelement *289*
TStatusBar Datenelement *489*
BORLANDC\BIN Verzeichnis
und DLLs *47, 48, 49*
Botschaft
WM_OCEVENT *783, 784*
Botschaften
Antwort *283*
Exception-Konstanten *68*
Senden *217*
Verarbeiten *215*
Verarbeitung *537*
Verarbeitung eingehender *567*
Vorverarbeiten *210, 217*
Botschaften senden *535*
Botschafts-Konstanten
MB_Xxxx *76*
Botschaftsbehandlungsfunktionen *583, 585, 587, 589*
Botschaftsextrahierroutinen *583, 585, 587, 589*
Botschaftskonstanten
Listenfenster *75*
Botschaftsverarbeitung
Fenster *531*
Botschaftszuteilungsroutine *83*
BottomLeft
TRect Elementfunktion *809*
BottomRight
TRect Elementfunktion *809*
Bounds
TGadget Datenelement *288*
BreakMessageLoop
TApplication Datenelement *90*
BringWindowToTop
TWindow Elementfunktion *534*
Browse Clipboard Elementfunktion
TOcApp *697*
Browse Elementfunktion
TOcApp *696*
BrowseClipboard Elementfunktion
TOcView *756*
BrowseLinks Elementfunktion
TOcView *756*
BS_DEFPUSHBUTTON Konstante *107*
BS_PUSHBUTTON Konstante *107*
BS_RADIOBUTTON-Konstante *453*
buffer
TInputDialog Datenelement *314*
BufferSize
TInputDialog Datenelement *314*
BuffSize
TFindReplaceDialog::TData Datenelement *273*
BuildCelArray
TButtonGadget Elementfunktion *112*
Button-Tastenargumente
VBX-Elemente *524*
BWCC-Templates *213*
BWCCEnabled
TApplication Elementfunktion *86*

Bytes
Größe der übertragenen Daten *484*, *485*

C

CalcPlaySize
TMetaFilePict Elementfunktion *365*
Callback-Funktionen *694*
CancelPressed
TButtonGadget Elementfunktion *112*
CanClear
TEditFile Elementfunktion *255*
CanClose
TApplication Elementfunktion *86*
TDocument Elementfunktion *228*
TEdit Elementfunktion *252*
TEditFile Elementfunktion *256*
TEditView Elementfunktion *260*
TListView Elementfunktion *332*
TOleDocument Elementfunktion, öffentliche *375*
TOleFrame Elementfunktion, geschützte *382*
TOleView Elementfunktion, geschützte *389*
TOleWindow Elementfunktion, geschützte *396*
TWindow Elementfunktion *534*
TWindowView Elementfunktion *575*
CanClose Elementfunktion
TOcApp *697*
CanDirty Datenelement
TOcVerb *754*
CanUndo
TEdit Elementfunktion *246*
CanUnload Elementfunktion
TOcRegistrar *744*
TRegistrar *768*
cApp Datenelement
TOcView *762*
CaptionFont
TTinyCaption Datenelement *500*
CaptionHeight
TTinyCaption Datenelement *500*
Capture
TGadgetWindow Datenelement *296*
CaretRect
TSlider Elementfunktion *477*
CascadeChildren
TMDIClient Elementfunktion *342*
catch-Schlüsselwort
und Exceptions *579*
CBN_xxxx
Kombinationsfenster-Botschafts-Konstanten *57*
CBS_AUTOHSCROLL Konstante *139*
CBS_DROPDOWN Konstante *139*
CBS_DROPDOWNLIST Konstante *139*
CBS_OWNERDRAWFIXED Konstante *139*
CBS_OWNERDRAWVARIABLE Konstante *139*
CBS_SIMPLE Konstante *139*
CBS_SORT Konstante *139*
cc
TChooseColorDialog Datenelement *122*
CDTitle
TCommonDialog Datenelement *152*
CeEditConvert
TOleWindow Elementfunktion, geschützte *396*
CeEditCopy
TOleWindow Elementfunktion, geschützte *396*
CeEditCut
TOleWindow Elementfunktion, geschützte *396*
CeEditDelete
TOleWindow Elementfunktion, geschützte *396*
CeEditInsertObject
TOleWindow Elementfunktion, geschützte *397*
CeEditLinks
TOleWindow Elementfunktion, geschützte *397*
CeEditObject
TOleWindow Elementfunktion, geschützte *397*
CeEditPaste
TOleWindow Elementfunktion, geschützte *397*
CeEditPasteLink
TOleWindow Elementfunktion, geschützte *397*
CeEditPasteSpecial
TOleWindow Elementfunktion, geschützte *397*
CeEditVerbs
TOleWindow Elementfunktion, geschützte *397*

CeFileClose
TOleWindow Elementfunktion, geschützte *397*
CelArray
TButtonGadget Datenelement *111*
CelOffset
TCelArray Elementfunktion *116*
CelRect
TCelArray Elementfunktion *116*
CelSize
TCelArray Elementfunktion *116*
cf
TChooseFontDialog Datenelement *124*
ChangeModeToPal
TDib Elementfunktion *221*
ChangeModeToRGB
TDib Elementfunktion *221*
char far*() operator *815*
char*() Elementfunktion
TAutoString *686*
Check
TCheckBox Elementfunktion *119*
Check Elementfunktion
TXOle *779*
TXRegistry *780*
Check State
TCommandEnabler Typdefinition *149*
Checkmarken *56*
CheckMenuItem
TMenu Elementfunktion *350*
CheckRadioButton
TWindow Elementfunktion *535*
CheckValid
TDC Elementfunktion *208*
TGdiObject Elementfunktion *305*
TMenu Elementfunktion *350*
ChildBroadcastMessage
TWindow Elementfunktion *535*
ChildDoc
TDocument Datenelement *227*
ChildWindowFromPoint
TWindow Elementfunktion *535*
ChildWithId
TWindow Elementfunktion *536*
Chord
TDC Elementfunktion *164*
ClassExpert
Dokumenteigenschaften *62*
Kommandobenachrichtigungen *57*, *58*, *59*
ClassInfo Datenelement
TAutoBool Struktur *660*
TAutoCurrency *665*
TAutoDate *665*
TAutoDouble *666*
TAutoFloat *669*
TAutoLong *673*
TAutoShort *681*
TAutoString *686*
TAutoVoid *693*
CleanupWindow
TOleFrame Elementfunktion, geschützte *382*
TOleWindow Elementfunktion, geschützte *398*
TWindow Elementfunktion *569*
Clear
TComboBox Elementfunktion *140*, *144*
TEdit Elementfunktion *246*
TListBoxData Elementfunktion *331*
TStatic Elementfunktion *485*
Clear Elementfunktion
TAutoVal *689*
TOcNameList *728*
TOcPartCollection *738*
ClearContainerGroupCount
TMenuDescr Elementfunktion *360*
ClearDevice
TPrinter Elementfunktion *439*
ClearDevMode
TPrintDialog::TData Elementfunktion *436*
ClearDevNames
TPrintDialog::TData Elementfunktion *436*
ClearFlag
TDocTemplate Elementfunktion *795*
TWindow Elementfunktion *536*
ClearFlag Elementfunktion
TAutoCommand *661*
ClearList
TComboBox Elementfunktion *140*
TListBox Elementfunktion *324*
ClearModify
TEdit Elementfunktion *246*
ClearServerGroupCount
TMenuDescr Elementfunktion *360*
Client-Fenster *209*, *282*
Bewegen durch *278*
ClientAttr
TMDIClient Datenelement *342*
ClientSize
TLayoutWindow Datenelement *323*

ClientToScreen
TWindow Elementfunktion *536*
ClientWnd
TFrameWindow Datenelement *282*
Clip
TGadget Datenelement *285*
Clip Datenelement
TOcViewPaint *764*
Clip Elementfunktion
TOcApp *697*
Clone *819*
TDocTemplate Elementfunktion *795, 800*
TGdiObject::TXGdi Elementfunktion *308*
TModule::TXInvalidModule Elementfunkltion *374*
TXInvalidMainWindow öffentliche Elementfunktion *93*
TXMenu Öffentliche Elementfunktion *577*
TXOwl Elementfunktion *580*
TXWindow Elementfunktion *576*
Clone Elementfunktion *819*
Close
TDocument Elementfunktion *229*
TFileDocument Elementfunktion *266*
TMetaFileDC Elementfunktion *364*
TStorageDocument Elementfunktion *491*
Close Elementfunktion
TOcDocument *708*
TOcPart *730*
CloseBox
TTinyCaption Datenelement *500*
CloseChildren
TMDIClient Elementfunktion *342*
CloseClipboard
TClipboard Elementfunktion *129*
CloseThisFile
TFileDocument Elementfunktion *268*
CloseWindow
TDialog Elementfunktion *214*
TWindow Elementfunktion *536*
clsid Registrierschlüssel *630*
CM_EXIT Botschaft *536*
CM_FILESAVEAS Botschaft *256*
CM_xxxx Bearbeiten-Ersetzen-Konstanten *58*
CM_xxxx Edit View-Konstanten *58*
CM_xxxx Editierdateikonstanten *58*
CM_xxxx MDI-Konstanten *59*
CmArrangeIcons
TMDIClient Elementfunktion *343*
CmCancel
TDialog Elementfunktion *214*
TFindReplaceDialog Elementfunktion *272*
CmCascadeChildren
TMDIClient Elementfunktion *343*
CmCharsEnable
TEdit Elementfunktion *252*
CmChildActionEnable
TMDIClient Elementfunktion *344*
CmCloseChildren
TMDIClient Elementfunktion *344*
CmCreateChild
TMDIClient Elementfunktion *344*
cmdline Registrierschlüssel *631*
CmEditAdd
TListView Elementfunktion *333*
CMEditClear
TEdit Elementfunktion *252*
TListView Elementfunktion *333*
CmEditConvert
TOleWindow Elementfunktion, geschützte *398*
CmEditCopy
TEdit Elementfunktion *252*
TListView Elementfunktion *334*
TOleWindow Elementfunktion, geschützte *398*
CmEditCut
TEdit Elementfunktion *252*
TListView Elementfunktion *334*
TOleWindow Elementfunktion, geschützte *398*
CmEditDelete
TEdit Elementfunktion *252*
TListView Elementfunktion *334*
TOleWindow Elementfunktion, geschützte *398*
CmEditFind
TEditSearch Elementfunktion *258*
CmEditFindNext
TEditSearch Elementfunktion *258*
CmEditInsertObject
TOleWindow Elementfunktion, geschützte *398*
CmEditItem
TListView Elementfunktion *334*
CmEditLinks
TOleWindow Elementfunktion, geschützte *398*
CmEditPaste

TEdit Elementfunktion *252*
TListView Elementfunktion *334*
TOleWindow Elementfunktion, geschützte *398*
CmEditPasteLink
TOleWindow Elementfunktion, geschützte *398*
CmEditPasteSpecial
TOleWindow Elementfunktion, geschützte *398*
CmEditReplace
TEditSearch Elementfunktion *258*
CmEditUndo
TEdit Elementfunktion *253*
TListView Elementfunktion *334*
CmExit
TWindow Elementfunktion *536*
CmFileClose
TDocManager Elementfunktion *237*
TOleWindow Elementfunktion, geschützte *399*
CmFileNew
TDocManager Elementfunktion *237*
TEditFile Elementfunktion *256*
CmFileOpen
TDocManager Elementfunktion *237*
TEditFile Elementfunktion *256*
CmFileRevert
TDocManager Elementfunktion *237*
CmFileSave
TDocManager Elementfunktion *237*
TEditFile Elementfunktion *256*
CmFileSaveAs
TDocManager Elementfunktion *237*
TEditFile Elementfunktion *256*
CmFindNext
TFindReplaceDialog Elementfunktion *272*
CmFontApply
TChooseFontDialog Elementfunktion *124*
CmHelp
TCommonDialog Elementfunktion *152*
CmLbSelChanged
TOpenSaveDialog Elementfunktion *410*
CmModEnable
TEdit Elementfunktion *253*
CmOk
TDialog Elementfunktion *214*
TOpenSaveDialog Elementfunktion *410*
CmOkCancel
TCommonDialog Elementfunktion *152*
CmPasteEnable
TEdit Elementfunktion *253*
CmReplace
TFindReplaceDialog Elementfunktion *272*
CmReplaceAll
TFindReplaceDialog Elementfunktion *272*
CmSelChange
TListView Elementfunktion *334*
CmSelectEnable
TEdit Elementfunktion *253*
CmSetup
TPrintDialog Elementfunktion *439*
CmTileChildren
TMDIClient Elementfunktion *344*
CmTileChildrenHoriz
TMDIClient Elementfunktion *344*
CmViewCreate
TDocManager Elementfunktion *238*
Code
ausnahmeunsicherer *556*
Color
TChooseColorDialog::TData Datenelement *123*
TChooseFontDialog::TData Datenelement *125*
COLORREF
typedef *836*
COM-Objekte
erzeugen *634*, *635*
CommandEnable
TButtonGadget Elementfunktion *110*
CommandEnable, TGadget
Elementfunktion *286*
CommDlgExtendedError-Codes *273*
Commit
TDocument Elementfunktion *229*
TFileDocument Elementfunktion *266*
TOleDocument Elementfunktion, öffentliche *376*
TStorageDocument Elementfunktion *491*
Compare
TLocaleString Elementfunktion *337*
CompareItem
TControl Elementfunktion *157*
COMPAREITEMSTRUCT
Struktur *836*
Condemn
TAppDictionary Elementfunktion *96*
TApplication Elementfunktion *86*
const char far*() Elementfunktion

TAutoString *686*
ConstructDoc
TDocTemplate Elementfunktion *795*
ConstructView
TDocTemplate Elementfunktion *795*
Container
eingebettete Objektklassen *728*, *755*
Contains
TRect Elementfunktion *809*
TRegion Elementfunktion *457*
Control
TControlGadget Datenelement *155*
Convert Elementfunktion
TOcApp *698*
Copies
TPrintDialog::TData Datenelement *435*, *436*
Copy
TEdit Elementfunktion *246*
Copy Elementfunktion
TAutoIterator *670*
TAutoVal *689*
TOcRemView *746*
TOcView *757*
CopyCursor
TModule Elementfunktion *368*
CopyIcon
TModule Elementfunktion *368*
Count Elementfunktion
TOcNameList *728*
TOcPartCollection *738*
CountClipboardFormats
TClipboard Elementfunktion *129*
Create
TAutoFactory Elementfunktion *97*
TBitmap Elementfunktion *102*
TDialog Elementfunktion *214*
TEditView Elementfunktion *260*
TGadgetWindow Elementfunktion *297*
TListView Elementfunktion *332*
TMDIClient Elementfunktion *342*
TOleFactoryBase *379*
TPalette Elementfunktion *417*
TWindow Elementfunktion *536*
CreateAbortWindow
TPrinter Elementfunktion *440*
CreateAnyDoc
TDocManager Elementfunktion *238*
CreateAnyView
TDocManager Elementfunktion *238*
CreateApp
TAutoFactory Elementfunktion *98*
TOleFactoryBase Template *379*
CreateAutoObject Elementfunktion
TRegistrar *768*
CreateCaret
TWindow Elementfunktion *536*
CreateChild
TMDIClient Elementfunktion *342*
CreateChildren
TWindow Elementfunktion *536*
CreateDialogParam Funktion *214*
CreateDoc
TDocManager Elementfunktion *238*
TDocTemplate Elementfunktion *795*, *800*
CreateObject
TOleFactoryBase Template *380*
CreateOcApp Elementfunktion
TOcRegistrar *743*
CreateOCView
TOleView Elementfunktion, geschützte *390*
TOleWindow Elementfunktion, geschützte *399*
CreateVerbPopup
TOleWindow Elementfunktion, geschützte *399*
CreateView
TDocManager Elementfunktion *238*
TDocTemplate Elementfunktion *796*
TDocTemplateT Elementfunktion *800*
CSize
TCelArray Datenelement *117*
CTL3D DLL *87*
CTL3D DLL Unterstützung *213*
Ctl3dEnabled
TApplication Elementfunktion *86*
Current Elementfunktion
TOcPartCollectionIter *740*
CurrentArg Datenelement
TAutoStack *683*
CurrentPreviewFont
TPrintPreviewDC Datenelement *449*
Cursor
Position, Einfügen von Text an der aktuellen *249*
Position, zurückgeben *248*, *250*
CursorModule
TWindow Datenelement *568*
CursorResId
TWindow Datenelement *568*

CustColors
TChooseColorDialog::TData Datenelement *123*
CustomFilter
TOpenSaveDialog::TData Datenelement *411*
Cut
TEdit Elementfunktion *247*

D

Data
TChooseColorDialog Datenelement *122*
TChooseFontDialog Datenelement *124*
TFindReplaceDialog Datenelement *272*
TOpenSaveDialog Datenelement *409*
TPrintDialog Datenelement *438*
TPrinter Datenelement *440*
Date Datenelement
TAutoDate *666*
Datei-Handles *268*
Dateien
Ansehen *265*
Dialogelement-IDs *66*
ID-Konstanten *68*
Öffnen *78, 268*
Rückgabe der Information an *68*
Sharing-Modi *82*
Speichern *269*
Verwalten *255, 268, 269*
Dateifehler
Öffnen einer Datei *268*
Dateiformate
Registrierung *646*
Dateinamenerweiterung
Registrierung *645*
Dateipuffer *265*
Daten
Einträge prüfen *511, 512, 513*
Einträge überprüfen *495*
Manipulieren *127*
nur numerische Einträge prüfen *454*
Speichern *78*
Transfermechanismus *157*
Zwischenablage, Abrufen *129*
Datenelement
Beispiel *54, 55*
Datenelemente
TOleFrame *385*
TOleWindow *394*
Datenelemente, geschützte
TGadget Klasse *288*
Datenelemente, öffentliche
TGadget Klasse *285*
Datenelementes *819*
Datenformate
Registrierung *654*
Datentransfer *455*
Datentypen
Automatisierung *603*
Datenübertragungsmedien
Registrierung *654*
DC
TChooseFontDialog::TData Datenelement *125*
TPrintout Datenelement *445*
DC Datenelement
TOcViewPaint *764*
dd Elementfunktion
Siehe auch TOcFormatList
Deaktivieren CTL3D DLL *87*
debugclsid Registrierschlüssel *631*
debugdesc Registrierschlüssel *632*
Debuggen
Server *631*
debugger Registrierschlüssel *632*
debugprogid Registrierschlüssel *633*
DECLARE_AUTOCLASS Makro *633*
DECLARE_COMBASESn Makros *634*
DECLARE_RESPONSE_TABLE Makro
TChooseColorDialog Klasse *123*
TClipboardViewer Klasse *134*
DECLARE_RESPONSE_TABLE-Makro *59*
DeepCopy
TMenu Elementfunktion *354*
DefaultProc
TWindow Datenelement *531*
DefaultProcessing
TWindow Elementfunktion *537*
DefaultProtocol
TClipboard Datenelement *129*
DefExt
TOpenSaveDialog::TData Datenelement *411*
DEFINE_APP_DICTIONARY-Makro *60*
DEFINE_AUTOAGGREGATE Makros *635*
DEFINE_AUTOCLASS Makro *635*
DEFINE_COMBASESn Makro *635*

DEFINE_DOC_TEMPLATE_CLASS-Makro *60*
DEFINE_RESPONSE_TABLE Makro
TDocManager Klasse *242*
DEFINE_RESPONSE_TABLE-Makros *60*
DEFINE_RESPONSE_TABLE1 Makro
TChooseFontDialog Klasse *125*
DEFINE_RESPONSE_TABLE2 Makro
TDecoratedFrame Klasse *211*
DefWindowProc
TDecoratedMDIFrame Elementfunktion *212*
TMDIFrame Elementfunktion *347*
TOleMDIFrame Elementfunktion, geschützte *387*
TWindow Elementfunktion *538*
Deklarationsbezeichner
Makro _ICLASS *613*
Makro _IFUNC *613*
Makros _OCFxxxx *613*
Delete Elementfunktion
TOcPart *730*
DeleteCondemned
TAppDictionary Elementfunktion *96*
DeleteItem
TControl Elementfunktion *157*
DELETEITEMSTRUCT
Struktur *837*
DeleteLine
TEdit Elementfunktion *247*
DeleteMenu
TMenu Elementfunktion *350*
DeleteSelection
TEdit Elementfunktion *247*
DeleteString
TComboBox Elementfunktion *140*
TListBox Elementfunktion *324*
DeleteSubText
TEdit Elementfunktion *247*
DeleteTemplate
TDocManager Elementfunktion *238*
DeltaPos
TScrollBar Elementfunktion *464*
description Registrierschlüssel *636*
Destroy
TDialog Elementfunktion *215*
TDocument::List Elementfunktion *234*
TOleFrame Elementfunktion, geschützte *382*
TSortedStringArray Elementfunktion *482*
TWindow Elementfunktion *538*
DestroyApp
TAutoFactory Elementfunktion *98*
TOleFactoryBase Template *380*
DestroyCaret
TWindow Elementfunktion *538*
DestroyStashedPopups
TOleFrame Elementfunktion, geschützte *382*
Destructor *819*
Destruktor
Beispiel *54*
TAppDictionary Klasse *95*
TAutoBase *660*
TAutoCommand *661*
TAutoProxy *678*
TAutoStack *682*
TAutoString *685*
TFileDocument Klasse *265*
TFindReplaceDialog Klasse *273*
TOcApp *704*
TOcDocument *708*
TOcNameList *727*
TOcPart *737*
TOcPartCollection *737*
TOcRegistrar *742*
TOcScaleFactor *750*
TOcView *761*
TOleAllocator *765*
TUnknown *775*
TXOle *779*
TXOwl Klasse *580*
Destruktoren
TApplication Klasse *85*
TBitmapGadget Klasse *102*
TCelArray Klasse *116*
TClipboard Klasse *129*
TComboBoxData Klasse *144*
TControlGadget Klasse *155*
TCreatedDC *159*
TCursor *161*
TDC *162*
TDialog Klasse *214*
TDib *220*
TDocTemplate Klasse *799*
TDocument Klasse *228*
TDocument::List Klasse *234*
TEditFile Klasse *255*
TEditView Klasse *260*
TFrameWindow Klasse *279*

TGdiObject Klasse *303*
TSlider Klasse *473*
TView Klasse *526*
TWindow Klasse *533*
TWindowDC *574*
TWindowView Klasse *574*
Detach
TSortedStringArray Elementfunktion *482*
Detach Elementfunktion
TOcNameList *728*
TOcPart *730*
TOcPartCollection *738*
DetachStream
TDocument Datenelement *234*
DeviceCapabilities
TPrintDC Elementfunktion *425*
DEVMODE
Struktur *838*
DialogBoxParam Funktion *215*
Dialogelement-IDs
TInputDialog *66*
Dialogelemente *156*
Botschaftskonstanten *61*
ID *216*
Zeichnen *157, 158*
Dialogfenster
Auffinden von Text *270*
Ausführen *215, 216*
Botschaften, verarbeiten *215, 217*
Botschafts-Konstanten *76*
CTL3D DLL *213*
Dateiverwaltung *255, 268, 269*
Elemente *216, 217, 258*
erzeugen *121, 124, 215*
Initialisieren *123, 124, 214, 216, 273*
modale *212, 213*
nicht-modal *124*
nicht-modale *121, 212, 213, 270, 271, 460*
Oberflächenelemente *214*
Schließen *214, 215*
Suchen von Text *271*
DialogFunction
TChooseColorDialog Elementfunktion *122*
TChooseFontDialog Elementfunktion *125*
TDialog Elementfunktion *215*
TFindReplaceDialog Elementfunktion *272*
TOpenSaveDialog Elementfunktion *410*
TPrintDialog Elementfunktion *439*
DIB
Array mit Cels *116*
Dienstprogramm TYPEREAD *608*
Direction
TGadgetWindow Datenelement *296*
directory Registrierschlüssel *637*
DirectoryList
TComboBox Elementfunktion *140*
TListBox Elementfunktion *324*
DirtyFlag
TDocument Datenelement *234*
TListView Datenelement *332*
DirtyLayout
TGadgetWindow Datenelement *296*
DisableAutoCreate
TWindow Elementfunktion *538*
DisableItem
TBitSet Elementfunktion *103*
DisableTransfer
TWindow Elementfunktion *539*
Dispatch
TEventHandler Elementfunktion *264*
TWindow Elementfunktion *539*
Dispatcher
TResponseTableEntry data member *460*
DispatchScroll
TWindow Elementfunktion *570*
DLGC_xxxx-Dialogelement-Botschaftskonst
anten *61*
DLLs
Erzeugen *79*
Exportieren *79*
Importieren *79*
dmxxxx-Verwaltungsmoduskonstanten *62*
dnxxxx Dokumentbotschafts-Konstanten
62
Doc
TStream Datenelement *495*
Doc:TView Datenelement *528*
docfilter Registrierschlüssel *637*
docflags Registrierschlüssel *637*
DoChangeCBChain
TClipboardViewer Elementfunktion *134*
DOCINFO
TPrintDC Datenelement *434*
DocList
TDocManager Datenelement *237*
DoCommand
TTinyCaption Elementfunktion *501*
DoCreate
TCommonDialog Elementfunktion *152*
TDialog Elementfunktion *215*

TFindDialog Elementfunktion *271*
TFindReplaceDialog Elementfunktion *272*
TReplaceDialog Elementfunktion *459*
DocTitleIndex
TFrameWindow Datenelement *282*
DoDestroy
TClipboardViewer Elementfunktion *134*
DoDrawClipboard
TClipboardViewer Elementfunktion *134*
DoExecute
TChooseColorDialog Elementfunktion *122*
TChooseFontDialog Elementfunktion *125*
TCommonDialog Elementfunktion *152*
TDialog Elementfunktion *215*
TFileOpenDialog Elementfunktion *269*
TFileSaveDialog Elementfunktion *269*
TOpenSaveDialog Elementfunktion *410*
TPrintDialog Elementfunktion *438*
Dokument
Benennung *528*
Dokumente
Botschaftskonstanten *62*
Eigenschaftsattribute *79*
Erzeugen *62*
Erzeugen von Templates *799*
Offen-Modi *78*
ohne assoziierte Dateien *227*
Schließen *62*
Schnittstellen *525, 526, 527*
Sharing-Modi, shxxxx Konstanten *82*
String-ID Konstanten *68*
Template-Konstanten *62*
Templates, Erzeugen *60*
Verwaltungsmoduskonstanten *62*
Viewing *525, 526, 527, 574*
Dokumentpfade und TFileDocument *267*
Dokumentverwaltung
Erzeugen *235*
DoLButtonUp
TTinyCaption Elementfunktion *501*
DoMouseMove
TTinyCaption Elementfunktion *501*
DoNCActivate
TTinyCaption Elementfunktion *501*
DoNCCalcSize
TTinyCaption Elementfunktion *502*
DoNCHitTest
TFloatingFrame Elementfunktionen *276*
TTinyCaption Elementfunktion *502*
DoNCLButtonDown
TTinyCaption Elementfunktion *502*
DoNCPaint
TTinyCaption Elementfunktion *502*
DoSearch
TEditSearch Elementfunktion *259*
DoSysCommand
TTinyCaption Elementfunktion *502*
DoSysMenu
TTinyCaption Elementfunktion *502*
double far*() Elementfunktion
TAutoVal *689*
double() Elementfunktion
TAutoVal *689*
DoVerb Elementfunktion
TOcPart *730*
DownHit
TTinyCaption Datenelement *500*
DPtoLP
TDC Elementfunktion *165*
Drag
TVbxControl Elementfunktion *515*
Drag Elementfunktion
TOcApp *698*
DragAcceptFiles
TWindow Elementfunktion *539*
DragFinish
TDropInfo Elementfunktion *801*
DragQueryFile
TDropInfo Elementfunktion *801*
DragQueryFileCount
TDropInfo Elementfunktion *802*
DragQueryFileNameLen
TDropInfo Elementfunktion *802*
DragQueryPoint
TDropInfo Elementfunktion *802*
Draw Elementfunktion
TOcPart *731*
DrawFocusRect
TDC Elementfunktion *165*
DrawIcon
TDC Elementfunktion *165*
DrawItem
TControl Elementfunktion *157*
TMenu Elementfunktion *351*
DRAWITEM Botschaft *157*
DrawMenuBar
TWindow Elementfunktion *539*
DrawText
TDC Elementfunktion *165*
Drucker

IDs *66*, *70*
Status, Bestimmen *70*
Ds
Oberflächenobjekte *545*
dtxxxx-Konstanten *62*
DynamicCast Funktion *638*

E

Editierfelder *245*, *258*
Attribute *246*, *250*
Benachrichtigungs-Codes *253*
erste sichtbare Zeile *247*
Formatierrechteck *248*, *250*
Handles *247*, *250*
Paßwort *248*, *250*
Registrierungs-Klassenname *254*
Rückgängig machen *246*, *247*, *251*
Schließen *252*
Tabulatorpositionen *251*
Text *245*, *246*, *247*, *248*, *249*, *250*, *251*, *252*, *253*, *254*
umgebrochene Zeilen *247*, *249*
Wortumbruch *251*
Editierfenster
Ressourcen und Assoziieren mit Objekten *246*
Eigenschaft
entfernen *557*
Eigenschaften
Editierfelder *245*
Kombinationsfenster *139*
Schalter *107*
Eigenschaftsattribute
pfxxxx-Konstanten *79*
Eigenschaftsindex und TFileDocument *266*
Einbenutzerbetrieb *781*
Eingabefelder *269*
Definieren der Zeichenmenge *270*
Prüfen auf ungültige Eingaben *270*
Suche nach ungültigen Einträgen *513*
Überprüfung auf ungültige Einträge *512*
Eingabefokus *278*
eingebetete Objekte
Container-Klassen *755*
eingebettete Objekte
Präsentations-Aspekte *652*, *655*
Elementfunktion
Inflate *810*
Elementfunktion, öffentliche
TCursor Klasse *161*
Elementfunktionen *54*, *55*, *814*, *819*
Definieren *153*
Ereignisbehandlung *64*
Kantenbedingungen *244*
TOleFrame *381*, *382*
TOleMDIFrame *387*
TOleView *389*
TOleWindow *393*, *396*
TSortedStringArray Klasse *482*
veraltet *216*
Zeiger *83*
Elementfunktionen, geschützte
TGadget Klasse *289*
Elementfunktionen, öffentliche
TGadget Klasse *286*
TOleDocument *375*
TOleFactoryBase *379*
Ellipse
TDC Elementfunktion *167*
Embedded
TDocument Datenelement *234*
EmptyClipboard
TClipboard Elementfunktion *129*
EmptyUndoBuffer
TEdit Elementfunktion *247*
EN_ERRSPACE Konstante *253*
EN_xxxx-Editierbotschaftskonstanten *64*
Enable
TButtonGadgetEnabler Elementfunktion *115*
TCommandEnabler Elementfunktion *150*
TMenuItemEnabler Elementfunktion *361*
EnableAutoCreate
TWindow Elementfunktion *539*
EnableBWCC
TApplication Elementfunktion *87*
EnableBWCC und TDialog *213*
EnableCtl3d
TApplication Elementfunktion *87*
EnableCtl3dAutosubclass
TApplication Elementfunktion *87*
EnableEditMenu Elementfunktion
TOcApp *698*
EnableItem
TBitSet Elementfunktion *103*
EnableKBHandler
TFrameWindow Elementfunktion *279*
EnableMenuItem

TMenu Elementfunktion *351*
EnableScrollBar
TWindow Elementfunktion *539*
EnableTinyCaption
TTinyCaption Elementfunktion *502*
EnableTransfer
TWindow Elementfunktion *539*
EnableWindow
TWindow Elementfunktion *540*
END_AUTOAGGREGATE Makro *638*
END_AUTOCLASS Makro *639*
END_RESPONSE_TABLE-Makro *63*
EndDialog Funktion *215*
EndDoc
TPrintDC Elementfunktion *428*
EndDocument
TPrintout Elementfunktion *443*
EndModal
TApplication Elementfunktion *87*
EndPage
TPrintDC Elementfunktion *428*
EndPath
TDC Elementfunktion *167*
EndPrinting
TPrintout Elementfunktion *443*
EndView
TScroller Elementfunktion *469*
ENErrSpace
TEdit Elementfunktion *253*
Entry
TEventInfo Datenelement *263*
EnumFontFamilies
TDC Elementfunktion *167*
EnumFonts
TDC Elementfunktion *168*
EnumMetaFile
TDC Elementfunktion *168*
EnumObjects
TDC Elementfunktion *168*
EnumProps
TWindow Elementfunktion *540*
EnumVerbs Elementfunktion
TOcPart *731*
Ereignis-IDs
und Views *581, 582*
Ereignisbehandlung
Suchen von Ereignissen *262*
Ereignisbehandlungsfunktionen *583, 585, 587, 589*
Ereignisbehandlungsroutinen *650*
Ereignisbotschaften
ObjectComponents *650, 651*
Ereigniskonstanten *263*
Ereignisse
Behandlung *59, 64*
Non-Client- *499, 500, 501, 502, 503, 504, 505*
Ereignistabellen *227*
Ereignisversendung
untergeordnete Fenster *521*
VBX-Elemente und *521*
Ermitteln von Informationen über
Ereignisse *262*
Error
TChooseColorDialog::TData Datenelement *123*
TChooseFontDialog::TData Datenelement *125*
TFilterValidator Elementfunktion *270*
TFindReplaceDialog::TData Datenelement *273*
TModule Elementfunktion *368*
TOpenSaveDialog::TData Datenelement *411*
TPrinter Datenelement *440*
TPXPictureValidator Elementfunktion *451*
TStringLookupValidator Elementfunktion *496*
TValidator Elementfunktion *511*
ErrorCode Datenelement
TXAuto *776*
ErrorCode Elementfunktion
TXObjComp *777*
Erzeugen von DCs über TCreateDC *162*
Erzeugen von HDCs *159*
Escape
TPrintDC Elementfunktion *428*
esComplete *263*
esPartial *263*
etActiveView Elementfunktion
Siehe TOcDocument
etName Elementfunktion
Siehe TOcDocument
etStorage Elementfunktion
Siehe TOcDocument
EV_CHILD_NOTIFY *64*
EV_CHILD_NOTIFY_ALL_CODES *64*
EV_CHILD_NOTIFY_AT_CHILD *65*
EV_COMMAND_AND_ID *65*
EV_MESSAGE *65, 385*

EV_OC_APPBORDERSPACEREQ *386*
EV_OC_APPBORDERSPACESET *386*
EV_OC_APPFRAMERECT *386*
EV_OC_APPINSMENUS *385, 388*
EV_OC_APPMENUS *385*
EV_OC_APPRESTOREUI *386*
EV_OC_APPROCESSMSG *386*
EV_OC_APPSHUTDOWN *386*
EV_OC_APPSTATUSTEXT *386*
EV_OWLDOCUMENT *65*
EV_OWLNOTIFY *65*
EV_OWLVIEW *65*
EV_REGISTERED *65*
EV_VBXEVENTINDEX *521*
EV_VBXEVENTNAME *522*
EV_VIEWNOTIFY *65*
EV_WM_ACTIVATEAPP *385, 388*
EV_WM_SIZE *385*
EV_xxxx-Makros *64*
EvActivate Elementfunktion
  TOcApp *699*
  TOcView *757*
EvActivateApp
  TOleFrame Elementfunktion, geschützte *382*
  TOleMDIFrame Elementfunktion, geschützte *387*
EvCanClose
  TDocManager Elementfunktion *238*
EvChangeCBChain
  TClipboardViewer Elementfunktion *134*
EvChar
  TEdit Elementfunktion *253*
EvChildInvalid
  TWindow Elementfunktion *540*
EvClose
  TCommonDialog Elementfunktion *152*
  TDialog Elementfunktion *215*
EvClose Botschaft *215*
EvClose Elementfunktion
  TOcRemView *746*
  TOcView *757*
EvCommand
  TDecoratedFrame Elementfunktion *210*
  TFrameWindow Elementfunktion *282*
  TOleWindow Elementfunktion, geschützte *399*
  TPrinterAbortDlg Elementfunktion *442*
  TTinyCaption Elementfunktion *503*
  TWindow Elementfunktion *540*
EvCommandEnable
  TDecoratedFrame Elementfunktion *210*
  TFrameWindow Elementfunktion *282*
  TOleWindow Elementfunktion, geschützte *399*
  TWindow Elementfunktion *540*
EvCtlColor
  TDialog Elementfunktion *215*
EvCtlColor Botschaft *215*
EvDestroy
  TClipboardViewer Elementfunktion *134*
EvDoVerb
  TOleWindow Elementfunktion, geschützte *400*
EvDrawClipboard
  TClipboardViewer Elementfunktion *134*
EvEnterIdle
  TDecoratedFrame Elementfunktion *210*
EvEraseBkgnd
  TFrameWindow Elementfunktion *282*
  TGauge Elementfunktion *301*
  TSlider Elementfunktion *473*
EvFindMsg
  TEditSearch Elementfunktion *259*
EvGetDlgCode
  TButton Elementfunktion *108*
  TCheckBox Elementfunktion *120*
  TEdit Elementfunktion *253*
  TListView Elementfunktion *334*
  TSlider Elementfunktion *474*
EvHScroll
  TScrollBar Elementfunktion *464*
EvInitDialog
  TDialog Elementfunktion *216*
EvInitDialog Botschaft *216*
EvInitMenuPopup
  TFrameWindow Elementfunktion *282*
EvKeyDown
  TEdit Elementfunktion *254*
  TSlider Elementfunktion *474*
EvKillFocus
  TEdit Elementfunktion *254*
  TSlider Elementfunktion *474*
EvLButtonDblClk
  TOleWindow Elementfunktion, geschützte *400*
  TSlider Elementfunktion *474*
EvLButtonDown
  TGadgetWindow Elementfunktion *297*

TOleWindow Elementfunktion, geschützte *400*
TSlider Elementfunktion *475*
EvLButtonUp
TGadgetWindow Elementfunktion *298*
TOleWindow Elementfunktion, geschützte *400*
TSlider Elementfunktion *475*
TTinyCaption Elementfunktion *503*
EvMDIActivate
TOleWindow Elementfunktion, geschützte *400*
EvMDICreate
TMDIClient Elementfunktion *344*
EvMDIDestroy
TMDIClient Elementfunktion *344*
EvMenuSelect
TDecoratedFrame Elementfunktion *210*
EvMouseMove
TGadgetWindow Elementfunktion *298*
TOleWindow Elementfunktion, geschützte *400*
TSlider Elementfunktion *475*
EvMouseMoveTTonyCaption Elementfunktion *503*
EvNCActivate
TTinyCaption Elementfunktion *503*
EvNCCalcSize
TTinyCaption Elementfunktion *503*
EvNCDestroy
TEditView Elementfunktion *261*
TFindReplaceDialog Elementfunktion *272*
EvNCHitTest
TTinyCaption Elementfunktion *503*
EvNCLButtonDown
TTinyCAption Elementfunktion *504*
EvNCPaint
TTinyCaption Elementfunktion *504*
EvOcAppBorderSpaceReq
TOleFrame Elementfunktion, geschützte *383*
EvOcAppBorderSpaceSet
TOleFrame Elementfunktion, geschützte *383*
EvOcAppDialogHelp
TOleFrame Elementfunktion, geschützte *383*
EvOcAppFrameRect
TOleFrame Elementfunktion, geschützte *383*
EvOcAppInsMenus
TOleFrame Elementfunktion, geschützte *383*
TOleMDIFrame Elementfunktion, geschützte *387*
EvOcAppMenus
TOleFrame Elementfunktion, geschützte *383*
EvOcAppProcessMsg
TOleFrame Elementfunktion, geschützte *383*
EvOcAppRestoreUI
TOleFrame Elementfunktion, geschützte *384*
EvOcAppShutdown
TOleFrame Elementfunktion, geschützte *384*
EvOcAppStatusText
TOleFrame Elementfunktion, geschützte *384*
EvOcEvent
TOleFrame Elementfunktion, geschützte *384*
TOleWindow Elementfunktion, geschützte *400*
EvOcPartInvalid
TOleWindow Elementfunktion, geschützte *401*
EvOcViewAttachWindow
TOleView Elementfunktion, geschützte *390*
TOleWindow Elementfunktion, geschützte *401*
EvOcViewBorderSpaceReq
TOleWindow Elementfunktion, geschützte *401*
EvOcViewBorderSpaceSet
TOleWindow Elementfunktion, geschützte *401*
EvOcViewClipData
TOleWindow Elementfunktion, geschützte *401*
EvOcViewClose
TOleView Elementfunktion, geschützte *390*
TOleWindow Elementfunktion, geschützte *402*
EvOcViewDrag
TOleWindow Elementfunktion, geschützte *402*
EvOcViewDrop

TOleWindow Elementfunktion, geschützte *402*
EvOcViewGetPalette
TOleWindow Elementfunktion, geschützte *402*
EvOcViewGetScale
TOleWindow Elementfunktion, geschützte *402*
EvOcViewGetSiteRect
TOleWindow Elementfunktion, geschützte *402*
EvOcViewInsMenus
TOleView Elementfunktion, geschützte *390*
TOleWindow Elementfunktion, geschützte *402*
EvOcViewLoadPart
TOleView Elementfunktion, geschützte *390*
TOleWindow Elementfunktion, geschützte *403*
EvOcViewOpenDoc
TOleView Elementfunktion, geschützte *390*
TOleWindow Elementfunktion, geschützte *403*
EvOcViewPaint
TOleWindow Elementfunktion, geschützte *403*
EvOcViewPartInvalid
TOleView Elementfunktion, geschützte *391*
EvOcViewPartSize
TOleWindow Elementfunktion, geschützte *403*
EvOcViewSavePart
TOleView Elementfunktion, geschützte *391*
TOleWindow Elementfunktion, geschützte *403*
EvOcViewScroll
TOleWindow Elementfunktion, geschützte *404*
EvOcViewSetScale
TOleWindow Elementfunktion, geschützte *404*
EvOcViewSetSiteRect
TOleWindow Elementfunktion, geschützte *404*
EvOcViewShowTools
TOleWindow Elementfunktion, geschützte *404*
EvOcViewTitle
TOleWindow Elementfunktion, geschützte *404*
EvPaint
TControl Elementfunktion *158*
TDialog Elementfunktion *216*
TFrameWindow Elementfunktion *283*
TSlider Elementfunktion *475*
EvParentNotify
TFrameWindow Elementfunktion *283*
EvPreProcessMenu
TDocManager Elementfunktion *239*
EvQueryDragIcon
TFrameWindow Elementfunktion *283*
EvRButtonDown
TOleWindow Elementfunktion, geschützte *404*
EvResize Elementfunktion
TOcApp *699*
TOcView *757*
EvSetCursor
TOleWindow Elementfunktion, geschützte *405*
EvSetFocus
TFrameWindow Elementfunktion *283*
TOleWindow Elementfunktion, geschützte *405*
TSlider Elementfunktion *475*
EvSetFocus Elementfunktion
TOcApp *699*
TOcView *758*
EvSetFont
TDialog Elementfunktion *216*
EvSetRGBColor
TChooseColorDialog Elementfunktion *122*
EvSize
TDecoratedFrame Elementfunktion *211*
TFrameWindow Elementfunktion *283*
TGadgetWindow Elementfunktion *298*
TLayoutWindow Elementfunktion *323*
TOleFrame Elementfunktion, geschützte *384*
TOleWindow Elementfunktion, geschützte *405*
TPreviewPage Elementfunktion *423*
TSlider Elementfunktion *475*
TStatic Elementfunktion *485*
EvSysColorChange
TGadgetWindow Elementfunktion *298*
EvSysCommand
TTinyCaption Elementfunktion *504*
TWindow Elementfunktion *540*
EvTimer

TOleFrame Elementfunktion, geschützte *384*
EvVbxDispatch
TVbxEventHandler Klasse *525*
EvVScroll
TScrollBar Elementfunktion *464*
EvWakeUp
TDocManager Elementfunktion *239*
Exception
TXInvalidMainWindow *91*
Verwendung des Schlüsselworts catch *579*
Verwendung des Schlüsselworts try *579*
Exception Bitflags
Aufzählungskonstanten xs *84*
Exception-Behandlung
TWindow *532*
unter Verwendung von ReceiveMessage *556*
Exception-Klassen
ObjectComponents *601*
Exceptionobjekte *819*
Exceptionobjekte auslösen *819*
Exceptions *818*
Botschaftskonstanten *68*
Fehler-Strings *76*
TValidator Klasse *514*
TWindow Klasse *576*
TXCompatibility Klasse *577*
TXInvalidWindow Klasse *92*
TXMenu Klasse *577*
TXOutOfMemory Klasse *577*
TXOwl Klasse *578, 579, 580*
ExcludeClipRect
TDC Elementfunktion *169*
ExcludeUpdateRgn
TDC Elementfunktion *169*
ExecDialog
TModule Elementfunktion *369*
ExecPrintDialog
TPrinter Elementfunktion *440*
Execute
TDialog Elementfunktion *216*
Execute Elementfunktion
TAutoCommand *662*
EXPOSE_APPLICATION Makro *639*
EXPOSE_DELEGATE Makro *640*
EXPOSE_INHERIT Makro *640*
EXPOSE_ITERATOR Makro *641*
EXPOSE_METHOD Makros *642*
EXPOSE_PROPxxxx Makros *643*
EXPOSE_QUIT Makro *644*
ExStyle
TWindowAttr-Struktur *529*
extension Registrierschlüssel *645*
Extent
TMetaFilePict Datenelement *366*
Extent Datenelement
TOcView *762*
ExtFloodFill
TDC Elementfunktion *169*
ExtractGroups
TMenuDescr Elementfunktion *360*
Extrahieren von Botschaften *583, 585, 587, 589*
Extrahieren von Windows-Botschaften *591*
ExtTextOut
TDC Elementfunktion *170*

F

Factory Klassen-Templates *71, 72, 73, 74*
Hierarchie *71*
Fail Elementfunktion
TAutoCommand *662*
far*() Elementfunktion
TAutoVal *692*
Farben
Auswählen *121*
Voreinstellung *217*
Farbzahl
NColors *78*
Fehler
TRangeValidator Elementfunktion *454*
Fehler-Codes
TFinReplaceDialog::TData Klasse *273*
Fehler-Strings *76*
Fehlerbotschaften
Maximale Zahl von Zeichen *76*
Fehlercodes
Automatisierungs-Server *627*
OLE *648*
Fenster
Bedingungen *242*
Benennung *92, 530, 532*
Bewegen durch *278*
Bewegen in einem *468*
Client *209, 278*
Dekorieren *209, 211*
erstellen *530ff*

Erzeugen *83, 91, 214*
Größe bestimmen *459*
Größenbestimmung *528*
Handles *130*
Hauptfenster *536*
Kette von Ansichten der Zwischenablage *133, 134*
Layout *459*
Layout-Bedingungen *317, 318, 319, 320, 321, 322*
neu anzeigen *555*
Plazieren *242*
Schließen *86, 214, 215, 536*
standardmäßige Botschaftsverarbeitung *531*
Standardprozeduren *531*
untergeordnete *536*
untergeordnetes *211, 214*
Zeichnen *283*
Fenster erzeugen
Größenbedingungen *317, 318, 319, 320, 321, 322*
Fensterstil-Konstanten *484*
FileData
TEditFile Datenelement *255*
filefmt Registrierschlüssel *646*
FileName
TEditFile Datenelement *255*
TOpenSaveDialog::TData Datenelement *412*
FillPath
TDC Elementfunktion *171*
FillRect
TDC Elementfunktion *171*
FillRgn
TDC Elementfunktion *171*
Filter
TOpenSaveDialog::TData Datenelement *412*
FilterIndex
TOpenSaveDialog::TData Datenelement *412*
Filterprüfungen *270*
Find
TApplication Elementfunktion *87*
TEventHandler Elementfunktion *264*
TSortedStringArray Elementfunktion *482*
Find Elementfunktion
TOcNameList *728*
TOcPartCollection *738*
FindChildMenu
TMDIFrame Elementfunktion *346*
FindColor
TDib Elementfunktion *221*
FindDocument
TDocManager Elementfunktion *239*
FindExactString
TListBox Elementfunktion *325*
FindIndex
TDib Elementfunktion *221*
FindProperty
TDocument Elementfunktion *229*
TFileDocument Elementfunktion *266*
TStorageDocument Elementfunktion *492*
TView Elementfunktion *526*
FindResource
TModule Elementfunktion *369*
FindString
TComboBox Elementfunktion *140*
TListBox Elementfunktion *325*
FindWhat
TFindReplaceDialog::TData Datenelement *274*
FirstGadget
TGadgetWindow Elementfunktion *293*
FirstThat
TSortedStringArray Elementfunktion *482*
TWindow Elementfunktion *541*
Flags
Oberflächenobjekte *561*
OLE-Ausführungsmodi *705*
TChooseColorDialog::TData Datenelement *123*
TChooseFontDialog::TData Datenelement *126*
TColor Elementfunktion *137*
TFindReplaceDialog::TData Datenelement *274*
TOpenSaveDialog::TData Datenelement *412*
FlashWindow
TWindow Elementfunktion *542*
FlattenPath
TDC Elementfunktion *171*
Fließende Rahmen und TFrameWindow *278*
float far*() Elementfunktion
TAutoVal *690*
float() Elementfunktion
TAutoVal *690*

FloodFill
TDC Elementfunktion *171*
Flush
TSortedStringArray Elementfunktion *482*
FlushDoc
TDocManager Elementfunktion *239*
Fokus
Umschalten *157, 158*
Font
TGadgetWindow Datenelement *296*
FontHeight
TGadgetWindow Datenelement *297*
FontType
TChooseFontDialog::TData Datenelement *126*
ForceAllBands
TPrintout Datenelement *445*
ForEach
TWindow Elementfunktion *542*
Formate
Registrierung *654*
Zwischenablage *653*
FormatLines
TEdit Elementfunktion *247*
FormatList Datenelement
TOcView *762*
formatn Registrierschlüssel *646*
ForwardEvent
TOcApp Elementfunktion *704*
ForwardEvent Elementfunktion
TOcView *761*
ForwardMessage
TWindow Elementfunktion *542, 543*
fr
TFindReplaceDialog Datenelement *272*
Frame
TTinyCaption Datenelement *500*
FrameRect
TDC Elementfunktion *172*
FrameRgn
TDC Elementfunktion *172*
Free Elementfunktion
TOleAllocator *766*
Funktion erstellen
abstrakt virtuelle *536*
Funktion GetApplicationObject *65*
Funktion GetWindowPtr *65*
Funktionen
Botschaftszuteilungsroutine *83*
Funktionskopf-Temmplate
im Antworttabellenmakro verwendet *591*
Funktionstyp
TActionFunc *542*
TActionMemFunc *542*
TCondFunc *153, 541*
TCondMemFunc *153*
Funktionstypen
TActionFunc *82*
TActionMemFunc *83*
TAnyAnyDispatcher *83*
TAnyPMF *83*

G

gadget
TButtonGadgetEnabler Datenelement *114*
GadgetChangedSize
TGadgetWindow Elementfunktion *293*
GadgetFromPoint
TGadgetWindow Elementfunktion *293*
GadgetReleaseCapture
TGadgetWindow Elementfunktion *293*
Gadgets *284, 285, 286, 287, 288, 289, 290*
TGadgetWindow Datenelement *297*
Trenner *471*
GadgetSetCapture
TGadgetWindow Elementfunktion *294*
GadgetWithId
TGadgetWindow Elementfunktion *294*
Geerbte Elementfunktionen *55*
Gekapselte Funktionen
GetParent *547*
GetProp *547, 562*
WIN API *821, 823, 825, 827*
Generischer Antworttabelleneintrag *262*
Geschützte Datenelement
TFileDocument Klasse *268*
Geschützte Datenelemente
TApplication Klasse *90*
TButton Klasse *107*
TCelArray Klasse *117*
TChooseFontDialog Klasse *124*
TClipboard Klasse *133*
TClipboardViewer Klasse *133*
TComboBoxData Klasse *145*
TCommandEnabler Klasse *150*
TCommonDialog Klasse *152*
TDecoratedFrame Klasse *210*
TDocument Klasse *234*

TEditView Klasse *261*
TFilterValidator Klasse *270*
TFindReplaceDialog Klasse *272*
TFrameWindow Klasse *282*
TGadgetWindow Klasse *296*
TGauge Klasse *301*
TLayoutWindow Klasse *323*
TListView Klasse *333*
TMemoryDC Klasse *349*
TMenu Klasse *354*
TMenuDescr Klasse *360*
TMenuItemEnabler Klasse *361*
TModule Klasse *373*
TOpenSaveDialog Klasse *409*
TPreviewPage Klasse *423*
TPrintDC Klasse *434*
TPrintDialog Klasse *438*
TPrinter Klasse *440*
TPrintout Klasse *445*
TPrintPreviewDC Klasse *449*
TPXPictureValidator Klasse *452*
TSlider Klasse *477*
TValidator Klasse *513*
TView Klasse *528*
TWindow Klasse *568*
Geschützte Datenelemente
TMessageBar Klasse *362*
Geschützte Datenelemente, TMetaFilePict
Klasse *366*
Geschützte Elementfunktionen
TApplication Klasse *91*
TBitmapGadget Klasse *102*
TButton Klasse *107*
TCheckBox Klasse *120*
TChooseColorDialog Klasse *122*
TChooseFontDialog Klasse *124*
TClipboardViewer Klasse *134*
TComboBox Klasse *143*
TCommonDialog Klasse *152*
TControl Klasse *158*
TDecoratedFrame Klasse *210*
TDecoratedMDIFrame Klasse *212*
TDialog Klasse *217*
TDocManager Klasse *241*
TDocument Klasse *234*
TEdit Klasse *252*
TEditFile Klasse *257*
TEditView Klasse *261*
TEventHandler Klasse *264*
TFileDocument Klasse *268*
TFindDialog Klasse *271*
TFindReplaceDialog Klasse *272*
TFloatingFrame Klasse *276*
TFrameWindow Klasse *282*
TGadgetWindow Klasse *297*
TGauge Klasse *301*
TInputDialog Klasse *315*
TLayoutWindow Klasse *323*
TListBox Klasse *329*
TListView Klasse *333*
TMDIChild Klasse *340*
TMDIClient Klasse *343*
TMDIFrame Klasse *347*
TMessageBar Klasse *363*
TOpenSaveDialog Klasse *410*
TPalette Klasse *417*
TPreviewPage Klasse *423*
TPrintDialog Klasse *439*
TPrinter Klasse *440*
TPrintPreviewDC Klasse *449*
TReplaceDialog Klasse *459*
TSlider Klasse *473*
TStatusBar Klasse *489*
TVbxControl Klasse *520*
TVbxEventHandler Klasse *525*
TView Klasse *528*
TWindow Klasse *569*
Geschützte Konstruktoren
TStream Klasse *495*
Geschützter Konstruktor
TWindow Klasse *569*
Geschütztes Datenelement
TEdit Klasse *252*
GetActiveMDIChild
TMDIClient Elementfunktion *343*
GetActivePart Elementfunktion
TOcView *758*
GetActiveView Elementfunktion
TOcDocument *708*
GetActiveWindow
TWindow Elementfunktion *543*
GetAppDescriptor Elementfunktion
TOcRegistrar *743*
GetApplication
TAppDictionary Elementfunktion *96*
TDocManager Elementfunktion *239*
TWindow Elementfunktion *543*
GetApplicationObject
Funktion *65*
GetAspectRatioFilter

TDC Elementfunktion *172*
GetAttributeHDC
TDC Elementfunktion *208*
TPrintPreviewDC Elementfunktion *449*
GetBitmapBits
TBitmap Elementfunktion *100*
GetBitmapDimension
TBitmap Elementfunktion *100*
GetBits
TDib Elementfunktion *221*
GetBkColor
TDC Elementfunktion *172*
TSlider Elementfunktion *475*
GetBkMode
TDC Elementfunktion *172*
GetBorders
TGadget Elementfunktion *286*
GetBorderStyle
TGadget Elementfunktion *286*
GetBoundingRect
TUIHandle Elementfunktion *510*
GetBounds
TGadget Elementfunktion *287*
GetBoundsRect
TDC Elementfunktion *172*
GetBrushOrg
TDC Elementfunktion *173*
GetBWCCModule
TApplication Elementfunktion *87*
GetCaptionRect
TTinyCaption Elementfunktion *504*
GetCapture
TWindow Elementfunktion *543*
GetCaretBlinkTime
TWindow Elementfunktion *543*
GetCaretIndex
TListBox Elementfunktion *325*
GetCaretPos
TWindow Elementfunktion *543*
GetCharABCWidths
TDC Elementfunktion *173*
GetCharWidth
TDC Elementfunktion *173*
GetCheck
TCheckBox Elementfunktion *119*
GetChildLayoutMetrics
TLayoutWindow Elementfunktion *322*
GetChildren
TWindow Elementfunktion *543*
GetClassInfo
TModule Elementfunktion *369*
GetClassLong
TWindow Elementfunktion *543*
GetClassName
TButton Elementfunktion *108*
TCheckBox Elementfunktion *121*
TComboBox Elementfunktion *143*
TDialog Elementfunktion *217*
TEdit Elementfunktion *254*
TGroupBox Elementfunktion *310*
TListBox Elementfunktion *329*
TMDIClient Elementfunktion *344*
TScrollBar Elementfunktion *466*
TStatic Elementfunktion *485*
TVbxControl Elementfunktion *520*
TWindow Elementfunktion *570*
GetClassWord
TWindow Elementfunktion *544*
GetClientHandle
TModule Elementfunktion *369*
GetClientRect
TWindow Elementfunktion *544*
GetClientWindow
TFrameWindow Elementfunktion *279*
TMDIFrame Elementfunktion *347*
GetClipboard
TClipboard Elementfunktion *129*
GetClipboardData
TClipboard Elementfunktion *129*
GetClipboardFormatName
TClipboard Elementfunktion *130*
GetClipboardOwner
TClipboard Elementfunktion *130*
GetClipboardViewer
TClipboard Elementfunktion *130*
GetClipBox
TDC Elementfunktion *173*
GetClipRgn
TDC Elementfunktion *174*
GetColor
TDib Elementfunktion *222*
GetColors
TDib Elementfunktion *222*
GetCommandTarget
TFrameWindow Elementfunktion *279*
TMDIFrame Elementfunktion *347*
GetContainerTitle Elementfunktion
TOcRemView *746*
GetCount
TComboBox Elementfunktion *140*

TListBox Elementfunktion *325*
GetCtl3dModule
TApplication Elementfunktion *87*
GetCurrentDoc
TDocManager Elementfunktion *239*
GetCurrentPosition
TDC Elementfunktion *174*
GetCursorId
TUIHandle Elementfunktion *510*
GetCursorPos
TWindow Elementfunktion *544*
GetDataType Elementfunktion
TAutoVal *690*
GetDCOrg
TDC Elementfunktion *174*
GetDefaultExt
TDocTemplate Elementfunktion *796*
GetDefaultId
TDialog Elementfunktion *216*
GetDefaultPrinter
TPrintDialog Elementfunktion *438*
TPrinter Elementfunktion *441*
GetDescription
TDocTemplate Elementfunktion *796*
GetDesiredSize
TBitmapGadget Elementfunktion *102*
TButtonGadget Elementfunktion *112*
TControlGadget Elementfunktion *155*
TGadget Elementfunktion *287*
TGadgetWindow Elementfunktion *298*
TMessagebar Elementfunktion *363*
TTextGadget Elementfunktion *498*
TToolBox Elementfunktion *506*
GetDesktopWindow
TWindow Elementfunktion *544*
GetDeviceCaps
TDC Elementfunktion *174*
TPrintPreviewDC Elementfunktion *446*
GetDeviceName
TPrintDialog::TData Elementfunktion *437*
GetDevMode
TPrintDialog::TData Elementfunktion *437*
GetDevNames
TPrintDialog::TData Elementfunktion *437*
GetDialogInfo
TPrintout Elementfunktion *443*
GetDIBits
TDC Elementfunktion *175*
GetDirection
TGadgetWindow Elementfunktion *294*
GetDirectory
TDocTemplate Elementfunktion *796*
GetDlgCtrlID
TWindow Elementfunktion *545*
GetDlgItem
TWindow Elementfunktion *545*
GetDlgItemInt
TWindow Elementfunktion *545*
GetDlgItemText
TWindow Elementfunktion *545*
GetDocManager
TApplication Elementfunktion *88*
TDocTemplate Elementfunktion *796*
TDocument Elementfunktion *229*
GetDocPath
TDocument Elementfunktion *229*
GetDocument
TStream Elementfunktion *495*
TView Elementfunktion *526*
GetDriverName
TPrintDialog::TData Elementfunktion *437*
GetDroppedControlRect
TComboBox Elementfunktion *140*
GetDroppedState
TComboBox Elementfunktion *140*
GetEditSel
TComboBox Elementfunktion *141*
GetEnabled
TGadget Elementfunktion *287*
GetErrorCode
TXOwl Elementfunktion *580*
GetEventIndex
TVbxControl Elementfunktion *515*
GetEventName
TVbxControl Elementfunktion *515*
GetExtendedUI
TComboBox Elementfunktion *141*
GetFactory Elementfunktion
TOcRegistrar *744*
TRegistrar *769*
GetFileFilter
TDocTemplate Elementfunktion *796*
GetFileTitle
TOpenSaveDialog Elementfunktion *409*
GetFileTitleLen
TOpenSaveDialog Elementfunktion *409*
GetFirstChild
TWindow Elementfunktion *545*
GetFirstVisibleLine
TEdit Elementfunktion *247*

GetFlags
TDocTemplate Elementfunktion *796*
GetFocus
TWindow Elementfunktion *545*
GetFont
TGadgetWindow Elementfunktion *294*
GetFontData
TDC Elementfunktion *175*
GetFontHeight
TGadgetWindow Elementfunktion *294*
GetGlyphDib
TButtonGadget Elementfunktion *112*
GetGlyphOutline
TDC Elementfunktion *175*
GetGroupCount
TMenuDescr Elementfunktion *359*
GetHandle
TEdit Elementfunktion *247*
TMenu Elementfunktion *351*
TMenuDescr Elementfunktion *359*
GetHandled
TCommandEnabler Elementfunktion *150*
GetHCTL
TVbxControl Elementfunktion *515*
GetHDC
TDC Elementfunktion *208*
GetHintMode
TGadgetWindow Elementfunktion *294*
GetHorizontalExtent
TListBox Elementfunktion *325*
GetHWndState
TWindow Elementfunktion *545*
GetIconInfo
TCursor Elementfunktion *161*
TIcon Elementfunktion *313*
GetId
TGadget Elementfunktion *287*
TMenuDescr Elementfunktion *359*
TWindow Elementfunktion *545*
GetIndex
TDib Elementfunktion *222*
GetIndices
TDib Elementfunktion *222*
GetInfo
TDib Elementfunktion *222*
GetInfoHeader
TDib Elementfunktion *222*
GetInitialRect Elementfunktion
TOcRemView *746*
GetInnerRect
TControlGadget Elementfunktion *155*
TGadget Elementfunktion *289*
TGadgetWindow Elementfunktion *298*
TMessagebar Elementfunktion *363*
GetInsertPosition
TOleWindow Elementfunktion, geschützte *405*
GetInstance
TModule Elementfunktion *370*
GetInstanceData
TModule Elementfunktion *370*
GetInt
TProfile Elementfunktion *450*
GetItemData
TComboBox Elementfunktion *141*
TListBox Elementfunktion *325*
GetItemDatas
TComboBox Elementfunktion *144*
TListBoxData Elementfunktion *331*
GetItemHandle
TDialog Elementfunktion *216*
GetItemHeight
TComboBox Elementfunktion *141*
TListBox Elementfunktion *325*
GetItemRect
TListBox Elementfunktion *326*
GetItemsInContainer
TSortedStringArray Elementfunktion *483*
GetKerningPairs
TDC Elementfunktion *176*
GetLastActivePopup
TWindow Elementfunktion *546*
GetLastChild
TWindow Elementfunktion *546*
GetLine
TEdit Elementfunktion *247*
GetLineFromPos
TEdit Elementfunktion *248*
GetLineIndex
TEdit Elementfunktion *248*
GetLineLength
TEdit Elementfunktion *248*
GetLogPerUnit
TOleWindow Elementfunktion, geschützte *405*
GetMainWindow
TApplication Elementfunktion *88*
GetMapMode
TDC Elementfunktion *176*
GetMargins

TGadget Elementfunktion *287*
TGadgetWindow Elementfunktion *299*
GetMaxBoxRect
TTinyCaption Elementfunktion *504*
GetMenu
TMenuItemEnabler Elementfunktion *361*
TWindow Elementfunktion *546*
GetMenuCheckMarkDimensions
TMenu Elementfunktion *351*
GetMenuDescr
TFrameWindow Elementfunktion *280*
GetMenuItemCount
TMenu Elementfunktion *351*
GetMenuItemID
TMenu Elementfunktion *351*
GetMenuState
TMenu Elementfunktion *351*
GetMenuString
TMenu Elementfunktion *352*
GetMetaFileBits
TMetaFilePict Elementfunktion *365*
GetMetaFileBitsEx
TMetaFilePict Elementfunktion *365*
GetMinBoxRect
TTinyCaption Elementfunktion *504*
GetModeIndicator
TStatusBar Elementfunktion *488*
GetModule
TDocTemplate Elementfunktion *796*
TMenuDescr Elementfunktion *359*
TWindow Elementfunktion *546*
GetModuleFileName
TModule Elementfunktion *370*
GetModuleUsage
TModule Elementfunktion *370*
GetName
TModule Elementfunktion *370*
GetName Elementfunktion
TOcApp *699*
TOcPart *731*
GetNameLen Elementfunktion
TOcPart *731*
GetNameList Elementfunktion
TOcApp *700*
GetNearestColor
TDC Elementfunktion *177*
GetNearestPaletteIndex
TPalette Elementfunktion *415*
GetNewStorage
TOleDocument Elementfunktion, öffentliche *376*
GetNextDlgGroupItem
TWindow Elementfunktion *546*
GetNextDlgTabItem
TWindow Elementfunktion *546*
GetNextTemplate
TDocManager Elementfunktion *240*
TDocTemplate Elementfunktion *796*
GetNextViewId
TView Elementfunktion *526*
GetNextWindow
TWindow Elementfunktion *546*
GetNumEntries
TPalette Elementfunktion *416*
GetNumEvents
TVbxControl Elementfunktion *516*
GetNumLines
TEdit Elementfunktion *248*
GetNumProps
TVbxControl Elementfunktion *516*
GetObject
TBitmap Elementfunktion *101*
TBrush Elementfunktion *106*
TFont Elementfunktion *277*
TGdiObject Elementfunktion *304*
TPalette Elementfunktion *416*
TPen Elementfunktion *420*
GetOcApp
TOleDocument Elementfunktion, öffentliche *376*
TOleFrame Elementfunktion, öffentliche *381*
TOleWindow Elementfunktion, öffentliche *393*
GetOcDoc
TOleDocument Elementfunktion, öffentliche *376*
TOleWindow Elementfunktion, öffentliche *393*
GetOcDocument Elementfunktion
TOcView *758*
GetOcRemView
TOleWindow Elementfunktion, öffentliche *394*
GetOcView
TOleWindow Elementfunktion, öffentliche *394*
GetOpenClipboardWindow
TClipboard Elementfunktion *130*

GetOpenMode
TDocument Elementfunktion *229*
TStream Elementfunktion *495*
GetOptions Elementfunktion
TRegistrar *769*
GetOrigin Elementfunktion
TOcView *758*
GetOuter Elementfunktion
TUnknown *773*
GetOuterSizes
TGadget Elementfunktion *287*
GetOutlineTextMetrics
TDC Elementfunktion *177*
GetOutputName
TPrintDialog::TData Elementfunktion *437*
GetPaletteEntries
TPalette Elementfunktion *416*
GetPaletteEntry
TPalette Elementfunktion *416*
GetParent
TWindow Elementfunktion *547*
GetParentDoc
TDocument Elementfunktion *230*
GetParentObject
TModule Elementfunktion *370*
GetPasswordChar
TEdit Elementfunktion *248*
GetPixel
TDC Elementfunktion *178*
GetPolyFillMode
TDC Elementfunktion *178*
GetPos Elementfunktion
TOcPart *732*
GetPosition
TMenuItemEnabler Elementfunktion *361*
TScrollBar Elementfunktion *464*
TSlider Elementfunktion *473*
GetPriorityClipboardFormat
TClipboard Elementfunktion *131*
GetProcAddress
TModule Elementfunktion *370*
GetProp
TVbxControl Elementfunktion *516, 517*
TWindow Elementfunktion *547*
GetProperty
TDocument Elementfunktion *230*
TFileDocument Elementfunktion *266*
TStorageDocument Elementfunktion *492*
TView Elementfunktion *527*
GetPropIndex
TVbxControl Elementfunktion *517*
GetPropName
TVbxControl Elementfunktion *517*
GetPropType
TVbxControl Elementfunktion *517*
GetRange
TGauge Elementfunktion *300*
TScrollBar Elementfunktion *464*
TSlider Elementfunktion *473*
GetRect
TEdit Elementfunktion *248*
GetRect Elementfunktion
TOcPart *732*
GetRefCount Elementfunktion
TUnknown *774*
GetRegistrar() Elementfunktion
TOcApp *700*
GetRegList
TDocTemplate Elementfunktion *797*
GetRemViewBucket
TOleFrame Elementfunktion, öffentliche *382*
GetRgnBox
TRegion Elementfunktion *457*
GetROP2()
TDC Elementfunktion *178*
GetScale Elementfunktion
TOcScaleFactor *751*
GetScaleFactor Elementfunktion
TOcScaleFactor *751*
GetScrollPos
TWindow Elementfunktion *547*
GetScrollRange
TWindow Elementfunktion *547*
GetSel
TListBox Elementfunktion *326*
GetSelCount
TComboBox Elementfunktion *144*
TListBox Elementfunktion *326*
GetSelection
TComboBox Elementfunktion *145*
TEdit Elementfunktion *249*
GetSelIndex
TComboBox Elementfunktion *141, 145*
TListBox Elementfunktion *326*
GetSelIndexes
TListBox Elementfunktion *326*
GetSelIndices
TListBoxData Elementfunktion *331*
GetSelString

TComboBox Elementfunktion *145*
TListBox Elementfunktion *326*
TListBoxData Elementfunktion *331*
GetSelStringLength
TComboBox Elementfunktion *145*
TListBoxData Elementfunktion *331*
GetSelStrings
TListBox Elementfunktion *326*
GetServerName Elementfunktion
TOcPart *732*
GetSize Elementfunktion
TOcPart *732*
GetState
TCheckBox Elementfunktion *119*
GetStorage
TStorageDocument Elementfunktion *492*
GetStreamName
TStream Elementfunktion *495*
GetStretchBltMode
TDC Elementfunktion *178*
GetString
TComboBox Elementfunktion *141*
TListBox Elementfunktion *327*
TProfile Elementfunktion *450*
GetStringLen
TComboBox Elementfunktion *141*
TListBox Elementfunktion *327*
GetStrings
TComboBox Elementfunktion *145*
TListBoxData Elementfunktion *331*
GetSubMenu
TMenu Elementfunktion *352*
GetSubText
TEdit Elementfunktion *249*
GetSymbol Elementfunktion
TAutoCommand *662*
TAutoIterator *670*
GetSysBoxRect
TTinyCaption Elementfunktion *504*
GetSysModalWindow
TWindow Elementfunktion *548*
GetSystemLangId
TLocaleString Elementfunktion *337*
GetSystemMenu
TWindow Elementfunktion *548*
GetSystemPaletteEntries
TDC Elementfunktion *178*
GetSystemPaletteUse
TDC Elementfunktion *178*
GetTabbedTextExtent
TDC Elementfunktion *178*
GetTemplate
TDocument Elementfunktion *230*
GetText
TComboBox Elementfunktion *142*
TStatic Elementfunktion *485*
TTextGadget Elementfunktion *497*
GetTextAlign
TDC Elementfunktion *179*
GetTextCharacterExtra
TDC Elementfunktion *180*
GetTextColor
TDC Elementfunktion *181*
GetTextExtent
TDC Elementfunktion *181*
GetTextFace
TDC Elementfunktion *181*
GetTextLen
TComboBox Elementfunktion *142*
TStatic Elementfunktion *485*
GetTextMetrics
TDC Elementfunktion *181*
GetThunk
TWindow Elementfunktion *548*
GetTitle
TDocument Elementfunktion *230*
TPrintout Elementfunktion *444*
GetTopIndex
TListBox Elementfunktion *327*
GetTopWindow
TWindow Elementfunktion *548*
GetType Elementfunktion
TAutoType *687*
GetUpdateRect
TWindow Elementfunktion *548*
GetUpdateRgn
TWindow Elementfunktion *548*
GetUserLangId
TLocaleString Elementfunktion *337*
GetValue
TGauge Elementfunktion *300*
GetVBXProperty
TVbxControl Elementfunktion *520*
GetViewId
TView Elementfunktion *527*
GetViewMenu
TView Elementfunktion *527*
GetViewName
TDocTemplate Elementfunktion *797*
TDocTemplateT Elementfunktion *801*

TEditView Elementfunktion *260*
TListView Elementfunktion *333*
TOleView Elementfunktion, öffentliche *389*
TView Elementfunktion *527*
TWindowView Elementfunktion *575*
GetViewportExt
TDC Elementfunktion *181*
GetViewportOrg
TDC Elementfunktion *182*
GetWindow
TEditView Elementfunktion *260*
TListView Elementfunktion *333*
TOleView Elementfunktion, öffentliche *389*
TView Elementfunktion *527*
TWindow Elementfunktion *548*
TWindowView Elementfunktion *575*
GetWindowClass
TDialog Elementfunktion *217*
TWindow Elementfunktion *570*
GetWindowExt
TDC Elementfunktion *182*
GetWindowFont
TWindow Elementfunktion *549*
GetWindowLong
TWindow Elementfunktion *549*
GetWindowOrg
TDC Elementfunktion *182*
GetWindowPlacement
TWindow Elementfunktion *549*
GetWindowPtr
Funktion *65*
TApplication Elementfunktion *88*
TWindow Elementfunktion *546*
GetWindowRect
TWindow Elementfunktion *549*
GetWindowRect Elementfunktion
TOcView *759*
GetWindowTask
TWindow Elementfunktion *549*
GetWindowText
TWindow Elementfunktion *550*
GetWindowTextLength
TWindow Elementfunktion *550*
GetWindowTextTitle
TWindow Elementfunktion *550*
GetWindowWord
TWindow Elementfunktion *550*
GetWinMainParams
TApplication Elementfunktion *88*
GetWordBreakProc
TEdit Elementfunktion *249*
Gleitregler
Abstände von Skalenpositionen *478*
Ausfüllen *475*
Beschreibung *472, 473, 474, 475, 476, 477, 478*
Bestimmen der Zeigerform *478*
Bewegen des Positionszeigers *477*
Hintergrundfarben *475*
Neuberechnung der Größe *475*
Objekte entfernen *473*
Objekte erstellen *473*
Reglerstellung *474*
Ressource-ID *478*
Rückgabe von Zeigerpositionen *477*
Setzen des Positionszeigers *475*
Speichern von Hintergrundfarben *477*
Übertragung von Zeigerpositionen *476*
vertikale *480*
zeichnen *474, 476*
Zeigerposition *472*
Zeigerpositionen *476*
globale Funktionen *65*
ObjectComponents *601*
Typumwandlung *638*
Zeiger zurückgeben *650*
Gray
TColor Datenelement *135*
GrayString
TDC Elementfunktion *182*
Green
TColor Elementfunktion *137*
Group
TCheckBox Datenelement *118*
GroupBox Klasse
Datenelemente, öffentliche *309*
Elementfunktionen, öffentliche *310*
Konstruktoren *309*
GroupCount
TMenuDescr Datenelement *360*
Gruppenfelder
Schaltfelder und *453*
GUID
Debug-Version *631*
Zuteilung *630*
GUIDGEN Dienstprogamm *630*
gültiges Zeichen
Eingabefelder *270*
Gültigkeitsprüfung *512*
Gültigkeitsüberprüfung *513*

H

H
  TDib Datenelement *225*
  TWindowAttr Struktur *530*
hAccel
  TWindow Datenelement *569*
HAccTable
  TApplication Datenelement *83*
Handle
  TDC Datenelement *207*
  TGdiObject Datenelement *305*
  TMenu Datenelement *354*
Handled
  TCommandEnabler Datenelement *150*
HandleMessage
  TWindow Elementfunktion *550*
handler Registrierschlüssel *647*
Handles
  Abfragen *130*
  Dialogfenster *216*
  Ermitteln *247*, *250*, *547*
Has
  TBitSet Elementfunktion *104*
HasActivePart
  TOleWindow Elementfunktion, öffentliche *394*
HasFocus
  TDocument Elementfunktion *230*
HasHScrollBar
  TScroller Datenelement *468*
HasMember
  TSortedStringArray Elementfunktion *483*
HasOption
  TValidator Elementfunktion *511*
HasPage
  TPrintout Elementfunktion *444*
HasVSCrollBar
  TScroller Datenelement *468*
Hatch11F1[8]
  THatch8x8Brush Datenelement *311*
Hatch13B1[8]
  THatch8x8Brush Datenelement *311*
Hatch13F1[8]
  THatch8x8Brush Datenelement *311*
Hatch22B1[8]
  THatch8x8Brush Datenelement *311*
Hatch22F1[8]
  THatch8x8Brush Datenelement *311*
Hatch8x8Brush Klasse
  Elementfunktion, öffentliche *312*
  Konstruktor *312*
Hauptfenster *548*
  Benennung *92*
  Erzeugen *83*, *91*
  nCmdShow anzeigen *84*
  Schließen *86*, *536*
HBottomTB Elementfunktion
  TOctool barInfo *752*
HBRUSH()
  TBrush Elementfunktion *106*
HCursor
  TWindow Datenelement *569*
HCURSOR()
  TCursor Operator *161*
Header-Dateien
  beschrieben *47*
  ObjectComponents *600*
  Übersicht *50*
Height
  TBitmap Elementfunktion *101*
  TDib Elementfunktion *222*
  TLayoutMetrics Datenelement *318*
  TMetaFilePict Elementfunktion *365*
  TRect Elementfunktion *809*
helpdir Registrierschlüssel *647*
HFILE_ERROR *265*
HFONT()
  TFont Elementfunktion *277*
HFrame Elementfunktion
  TOctool barInfo *752*
HideCaret
  TWindow Elementfunktion *551*
HideList
  TComboBox Elementfunktion *142*
Hierarchiediagrammm *3ff*
HighlightLine
  TMessagebar Datenelement *362*
HighValue
  TScrollBar Datenelement *467*
Hilfe-Schalter
  Reaktion auf Ereignisse *706*
Hilfedateien
  registrieren *781*
  Registrierung *647*
HiliteMenuItem
  TWindow Elementfunktion *551*
HInstance
  TModule Datenelement *373*
HintMode

TGadgetWindow Datenelement *297*
HintText
TMessagebar Datenelement *362*
HitTest
THSlider Elementfunktion *479*
TSlider Elementfunktion *476*
TUIHandle Elementfunktion *510*
TVSlider Elementfunktion *480*
HLeftTB Elementfunktion
TOctool barInfo *753*
HMENU
TMenuItemEnabler Datenelement *361*
HoldFocusHwnd
TFrameWindow Elementfunktion *280*
TWindow Elementfunktion *551*
HoldMenu
TOleFrame Datenelement, geschütztes *385*
Hook-Prozeduren für die Automatisierung ausführen *606*
Horizontale Bildlaufleiste
Umwandlung *471*
HPrevInstanz
TApplication Datenelement *84*
HR_xxxx Rückgabekonstanten *648*
HRightTB Elementfunktion
TOctool barInfo *753*
HScroll
TScroller Elementfunktion *470*
HTopTB Elementfunktion
TOctool barInfo *753*
HWindow
TWindow Datenelement *531*
HWND
TWindow Elementfunktion *551*
HWndNext
TClipboardViewer Datenelement *133*
HWndReceiver
TCommandEnabler Datenelement *150*
HWndRestoreFocus
TFrameWindow Datenelement *282*

I

I32_Dispatch *593*
I32_LPARAM_Dispatch *593*
I32_MenuChar_Dispatch *594*
I32_U_Dispatch *594*
I32_WPARAM_LPARAM_Dispatch *593*
i_LPARAM_Dispatch *593*
i_U_W_U_Dispatch *593*
i_WPARAM_Dispatch *593*
iconindex Registrierschlüssel *648*
Id
TCommandEnabler Datenelement *150*
TEventInfo Datenelement *263*
TGadget Datenelement *289*
TMenuDescr Datenelement *360*
TWindowAttr-Struktur *529*
ID_xxxx Dateikonstanten *66*
ID_xxxx Druckerkonstanten *66*
IDA_xxxx
OLE-Tastenkürzel-ID-Konstanten *66*
IDA_xxxx Tastenkürzel-ID-Konstanten *66*
IDCANCEL Konstante *214*
IDispatch&() Elementfunktion
TAutoProxy *679*
TAutoVal *690*
IDispatch*() Elementfunktion
TAutoProxy *679*
TAutoVal *690*
IdleAction
TApplication Elementfunktion *91*
TFrameWindow Elementfunktion *280*
TGadgetWindow Elementfunktion *294*
TStatusBar Elementfunktion *489*
TWindow Elementfunktion *551*
IDM_xxxx Menü-ID-Konstanten *67*
IDM_xxxx OLE-Menü-ID-Konstanten *67*
IDOK Konstante *214*
IDs
Vorgabewert ermitteln *216*
IDS_INVALIDMAINWINDOW
TXInvalidMainWindow Konstruktor *93*
IDS_xxxx Dokument String-ID Konstanten *68*
IDS_xxxx Drucker String-ID-Konstanten *70*
IDS_xxxx Edit-View-ID-Konstanten *67*
IDS_xxxx Editierdatei ID-Konstanten *68*
IDS_xxxx Exception-Botschaften *68*
IDS_xxxx List-View String-ID-Konstanten *69*
IDS_xxxx Modus-Konstanten *67*
IDS_xxxx Prüfungs-ID-Konstanten *70*
IDW_MDICLIENT Konstante *70*
IDW_MDIFIRSTCHILD Konstante *70*
Index
TColor Elementfunktion *138*
Inflate Elementfunktion *810*
Inflated

TRect Elementfunktion *810*
InflatedBy
TRect Elementfunktion *810*
Info
TDib Datenelement *225*
InfoFromHandle
TDib Elementfunktion *226*
Inhalt
Übersicht *27*
Init
TDC Elementfunktion *208*
TFindReplaceDialog Elementfunktion *273*
TFrameWindow Elementfunktion *283*
TOleWindow Elementfunktion, geschützte *405*
TOpenSaveDialog Elementfunktion *410*
TWindow Elementfunktion *572*
Init Elementfunktion
TAutoIterator *670*
TOcView *761*
InitApplication
TApplication Elementfunktion *91*
InitChild
TMDIClient Elementfunktion *343*
InitDoc
TDocManager Elementfunktion *240*
TDocTemplate Elementfunktion *797*
TDocument Elementfunktion *230*
TOleDocument Elementfunktion, öffentliche *376*
InitialDir
TOpenSaveDialog::TData Datenelement *413*
Initialisierung
Dialogfenster *214, 216*
Instanz *83*
InitInstance
TApplication Elementfunktion *91*
InitMainWindow
TApplication Elementfunktion *92*
InitModule
TModule Elementfunktion *370*
InitView
TDocTemplate Elementfunktion *797*
inline-Funktionen *29*
Insert
TDecoratedFrame Elementfunktion *209*
TDocument::List Elementfunktion *234*
TEdit Elementfunktion *249*
TGadgetWindow Elementfunktion *294*
TStatusBar Elementfunktion *488*
TToolBox Elementfunktion *506*
insertable Registrierschlüssel *649*
Inserted
TControlGadget Elementfunktion *155*
TGadget Elementfunktion *289*
InsertMenu
TMenu Elementfunktion *352, 353*
InsertString
TComboBox Elementfunktion *142*
TListBox Elementfunktion *327*
InstanceCount *819*
InstanceCount Datenelemente *819*
Instanzinitialisierung *83*
InStream
TDocument Elementfunktion *230*
TFileDocument Elementfunktion *266*
TStorageDocument Elementfunktion *492*
int
Typdefinition TSortedStringArray *481*
int far*() Elementfunktion
TAutoVal *690*
int() Elementfunktion
TAutoString *685*
TAutoVal *690*
Integerbereiche
durchsuchen *454*
IntersectClipRect
TDC Elementfunktion *183*
Invalidate
TButtonGadget Elementfunktion *113*
TControlGadget Elementfunktion *156*
TGadget Elementfunktion *289*
TTextGadget Elementfunktion *498*
TWindow Elementfunktion *551*
Invalidate Elementfunktion
TOcRemView *746*
InvalidatePart
TOleWindow Elementfunktion, geschützte *406*
InvalidatePart Elementfunktion
TOcView *759*
InvalidateRect
TControlGadget Elementfunktion *156*
TGadget Elementfunktion *290*
TWindow Elementfunktion *552*
InvalidateRgn
TWindow Elementfunktion *552*
InvertRect
TDC Elementfunktion *184*

InvertRgn
TDC Elementfunktion *184*
Invoke Elementfunktion
TAutoProxy *680*
InvokeElementfunktion
TAutoCommand *662*
IsActive Elementfunktion
TOcPart *733*
IsArrayProp
TVbxControl Elementfunktion *518*
IsAutoMode
TScroller Elementfunktion *470*
IsBound Elementfunktion
TAutoProxy *679*
IsChild
TWindow Elementfunktion *552*
IsClipboardFormatAvailable
TClipboard Elementfunktion *131*
IsCore
TDib Datenelement *225*
IsCurrentDefPB
TButton Datenelement *107*
IsDefPB
TButton Datenelement *107*
IsDirty
TDocument Elementfunktion *231*
IsDlgButtonChecked
TWindow Elementfunktion *552*
IsEmbedded
TDocument Elementfunktion *231*
IsEmpty
TBitSet Elementfunktion *104*
TRect Elementfunktion *810*
TSortedStringArray Elementfunktion *483*
IsEmpty Elementfunktion
TOcNameList *728*
TOcPartCollection *738*
IsFlagSet
TDocManager Elementfunktion *240*
TWindow Elementfunktion *552*
IsFlagSet TDocTemplate Elementfunktion
797
IsFull
TSortedStringArray Elementfunktion *483*
IsHorizontal
TGauge Datenelement *301*
IsIconic
TWindow Elementfunktion *553*
IsLink Elementfunktion
TOcPart *733*
IsLoaded
TModule Elementfunktion *370*
IsModal
TDialog Datenelement *213*
IsModified
TEdit Elementfunktion *249*
IsMyKindOfDoc
TDocTemplate Elementfunktion *797*
TDocTemplateT Elementfunktion *800*
IsMyKindOfView
TDocTemplate Elementfunktion *797*
TDocTemplateT Elementfunktion *801*
IsNativeLangId
TLocaleString Elementfunktion *337*
IsNull
TRect Elementfunktion *810*
IsOK
TDib Elementfunktion *222*
TMenu Elementfunktion *353*
TView Elementfunktion *527*
IsOpen
TClipboard Datenelement *133*
TDocument Elementfunktion *231*
TFileDocument Elementfunktion *266*
TStorageDocument Elementfunktion *492*
IsOpenEditing Elementfunktion
TOcRemView *747*
IsOptionSet Elementfunktion
TOcApp *700*
TRegistrar *769*
IsPM
TDib Elementfunktion *223*
IsPressed
TTinyCaption Datenelement *500*
IsPropSet Elementfunktion
TAutoCommand *662*
IsReceiver
TCommandEnabler Elementfunktion *150*
IsRef Elementfunktion
TAutoVal *691*
IsResHandle
TDib Datenelement *225*
IsStatic
TDocTemplate Elementfunktion *797*
IsString
TResID Elementfunktion *815*
IsValid
TEdit Elementfunktion *249*
TFilterValidator Elementfunktion *270*
TLookupValidator Elementfunktion *338*

TRangeValidator Elementfunktion *454*
TValidator Elementfunktion *511*
IsValidInput
TFilterValidator Elementfunktion *270*
TPXPictureValidator Elementfunktion *451*
TValidator Elementfunktion *512*
IsVisible
TDocTemplate Elementfunktion *798*
IsVisible Elementfunktion
TOcPart *733*
IsVisibleRect
TScroller Elementfunktion *470*
IsWindow
TWindow Elementfunktion *553*
IsWindowEnabled
TWindow Elementfunktion *553*
IsWindowVisible
TWindow Elementfunktion *553*
IsZoomed
TWindow Elementfunktion *553*
IsZoomed Elementfunktion
TOcScaleFactor *751*
ItemDatas
TComboBoxData Datenelement *145*
TListBoxData Datenelement *330*
Iterate
TAppDictionary Elementfunktion *96*
Iterator-Elementfunktionen
untergeordnete Fenster *541*
IUnknown Elementfunktion
TAutoIterator *671*
IUnknown&() Elementfunktion
TAutoVal *691*
IUnknown*() Elementfunktion
TAutoVal *691*

K

Kantenbedingungen, Erzeugen von
Fenstern *242*
Kapselung
Windows-API-Zwischenablagefunktionen *127*
Key Elementfunktion
TXRegistry *781*
KeyboardHandling
TFrameWindow Datenelement *279*
KillTimer
TWindow Elementfunktion *553*
Klänge, Piepton *253*
Klasse
TAutoBase *659*
TAutoCommand *660ff*
TAutoIterator *669ff*
TAutoProxy *677ff*
TAutoStack *681ff*
TAutoVal *687ff*
TOcApp *695ff*
TOcDocument *706ff*
TOcFormatList *713, 714*
TOcFormatListIter *715, 716*
TOcFormatName *717*
TOcInitInfo *718, 719, 720, 721*
TOcModule *724, 725*
TOcNameList *726, 727*
TOcPart *728ff*
TOcPartCollection *737, 738*
TOcPartCollectionIter *739, 740*
TOcRegistrar *741, 742, 743*
TOcRemView *744ff*
TOcScaleFactor *748ff*
TOcVerb *753, 754*
TOcView *755ff*
TOleAllocator *765*
TRegistrar *766ff*
TResId *814, 815*
TUnknown *772ff*
TXAuto *776*
TXBase *819*
TXObjComp *777*
TXOle *778, 779*
TXRegistry *780*
Klassen
Ereignisbehandlung *59*
Fenster-Registrierung *557*
inline-Funktionen *29*
Namen *217, 254*
TBIVbxLibrary *104*
TBrush *105*
TColor *135*
TDropInfo *801*
TPointer *806*
TRect *807ff*
TRegion *455ff*
TResId *815*
TRgbQuad *461*
TRgbTriple *461*
TSize *816, 817*
TXBase *818, 819*

TXBase *819*
Zugriffs-Spezifizierer *29*
Zugriffsoperator *29*
Klassen-Templates
Factory *71, 72, 73, 74*
Klassennamen
Registrierung von Oberflächenobjekten *570*
Kombinationsfenster *57*
durch übergeordnetes Fenster gezeichnet *157, 158*
Eigenschaften *139*
Einträge, Transferieren *143, 144*
Erzeugen *139*
Größe *158*
Listen *143*
Kombinationsfenster-Botschafts-Konstanten *57*
Kommando-basierte Botschaftskonstanten *57*
Komplementoperator in TBitSet verwendet *104*
Konstanten
Datentransfer *507*
Editierbotschaften *64*
Ereignis *263*
Exception-Botschaften *68*
Kombinationsfenster-Botschafts- *57*
Kommandobotschaften *57*
Registriertabellen *612*
Ressourcen-IDs *66, 67*
SB_Xxxx Bildlaufleisten- *82*
Schalter-Flags *56*
Schalterbotschafts- *56*
Transferfunktionen *507*
Konstructor *819*
Konstruktor
Beispiel *54, 55*
TAppDictionary Klasse *95*
TAutoCommand *661*
TAutoDate *666*
TAutoIterator *672*
TAutoProxy *680*
TAutoStack *682*
TButtonGadgetEnabler Klasse *114*
TDecoratedFrame Klasse *209*
TDesktopDC *212*
TDocTemplateT Elementfunktion *800*
TFindReplaceDialog Klasse *273*
TFloatingFrame Klasse *275*
TFrameWindow Klasse *282*
TOcApp *703*
TOcNameList *727*
TOcPartCollection *737*
TOcPartCollectionIter *740*
TOcRegistrar *742*
TOcRemView *745*
TOcVerb *754*
TOcView *755*
TOleAllocator *765*
TRangeValidator Klasse *454*
TRegistrar *767, 772*
TSortedStringArray Klasse *481*
TUIHandle Klasse *509*
TUnknown *775*
TXAuto *776*
TXObjComp *777*
TXOwl Klasse *580*
Konstruktoren
TApplication Klasse *85*
TAutoString *684*
TBitmap *99*
TBitmapGadget Klasse *102*
TBitSet Klasse *103*
TBIVbxLibrary Klasse *105*
TBrush *105*
TButton Klasse *107*
TButtonGadget Klasse *109*
TCelArray Klasse *115*
TCharSet Klasse *117*
TCheckBox Klasse *118*
TChooseColorDialog Klasse *121*
TChooseFontDialog Klasse *124*
TClientDC *127*
TClipboard Klasse *133*
TClipboardViewer Klasse *133, 134*
TColor *136*
TComboBox Klasse *139*
TComboBoxData Klasse *144*
TCommandEnabler Klasse *149*
TCommonDialog Klasse *151*
TControl Klasse *157*
TControlBar Klasse *154*
TControlGadget Klasse *155*
TCreatedDC *159*
TCursor *160*
TDC *162*
TDC Klasse *207*
TDecoratedMDIFrame Klasse *211*
TDialog Klasse *213*
TDib *220*

TDibDC *226*
TDocManager Klasse *236*
TDocTemplate Klasse *798*
TDocument Klasse *228*
TDocument::List Klasse *234*
TDropInfo *801*
TEdit Klasse *246*
TEditFile Klasse *255*
TEditSearch Klasse *258*
TEditView Klasse *259*
TEventInfo *263*
TFileDocument Klasse *265*
TFileOpenDialog Klasse *268*
TFileSaveDialog Klasse *269*
TFilterValidator Klasse *270*
TFindDialog Klasse *271*
TFindReplaceDialog Klasse *271*
TFont *276*
TFrameWindow Klasse *278, 279*
TGadgetKlasse *285*
TGadgetWindow Klasse *293*
TGadgetWindowFont Klasse *292*
TGauge Klasse *300*
TGdiObject Klasse *306*
TGdiObject::TXGdi Klasse *308*
TGroupBox Klasse *309*
THatch8x8Brush Klasse *312*
TIcon Klasse *313*
TInputDialog Klasse *314*
TInStream Klasse *315*
TLayoutMetrics Klasse *318*
TLayoutWindow Klasse *322*
TListBox Klasse *323*
TListBoxData Struktur *330*
TListView Klasse *332*
TLookupValidator Klasse *338*
TMDIChild Klasse *339*
TMDIClient Klasse *341*
TMDIFrame Klasse *346*
TMemoryDC Klasse *348*
TMenu Klasse *349*
TMenuDescr Klasse *358*
TMenuItemEnabler Klasse *361*
TMessageBar Klasse *362*
TMetaFileDC Klasse *363*
TMetaFilePict Klasse *364*
TModule Klasse *366*
TModule::TXInvalidModule Klasse *373*
TOcDocument *708*
TOcPart *729*
TOcScaleFactor *749*
TOleClientDC *374*
TOleDocument *375*
TOleFrame *381*
TOleMDIFrame *387*
TOleView *389*
TOleWindow *393*
TOpenSaveDialog Klasse *409*
TOutStream Klasse *413*
TPaintDC Klasse *414*
TPalette Klasse *414*
TPaletteEntry Klasse *418*
TPen Klasse *419*
TPopupMenu Klasse *421*
TPreviewPage Klasse *422*
TPrintDC Klasse *424*
TPrintDialog Klasse *438*
TPrinter Klasse *439*
TPrinter::TXPrinter Klasse *441*
TPrinterAbortDlg Klasse *442*
TPrintout Klasse *442*
TPrintPreviewDC Klasse *446*
TProfile Klasse *450*
TPXPictureValidator Klasse *451*
TRadioButton Klasse *453*
TRegion *455*
TReplaceDialog Klasse *459*
TRgbQuad *461*
TRgbTriple *462*
TScrollBar Klasse *463*
TScroller Klasse *469*
TSlider Klasse *472*
TView Klasse *526*
TWindow Klasse *533*
TWindowDC *574*
TWindowView Klasse *574*
TXOle *778*
TXRegistry *780*
TXWindow *576*
Konstruktoren TWindow Klasse *532*
Konstruktoren, TIC Klasse *312*
Konstruktoren, TStatic Klasse *484*
Konvertieren
Bereichswerte zu Bildlaufwerten *76*
Bildlaufwerte zu Bereichswerten *76*
Kopieren *819*
Kopieren von Exception-Objekten *93*
Kopierende Exceptionobjekte *819*

L

LangId Datenelement
TAutoStack *683*
language Registrierschlüssel *649*
LangXxxx ID-Konstanten *75*
LastThat
TSortedStringArray Elementfunktion *483*
Layout
TLayoutWindow Elementfunktion *322*
LayoutSession
TGadgetWindow Elementfunktion *295*
LayoutUnitsToPixels
TGadgetWindow Elementfunktion *299*
LBN_xxxx
Listenfensterbotschafts-Konstanten *75*
LButtonDown
TButtonGadget Elementfunktion *113*
TGadget Elementfunktion *290*
LButtonUp
TButtonGadget Elementfunktion *113*
TGadget Elementfunktion *290*
LedSpacing
TGauge Datenelement *301*
LedThick
TGauge Datenelement *301*
LeftOf
TEdgeConstraint Elementfunktion *243*
LineMagnitude
TScrollBar Datenelement *463*
LineTo
TDC Elementfunktion *184*
Link Datenelement
TOcView *762*
List-View-Konstanten *69*
Liste untergeordneter Fenster
Oberflächenelement und *572*
Listen untergeordneter Fenster
Oberflächenelement und *211, 284*
Objekte entfernen aus *562*
Transferpuffer und *569*
Listenfelder *158*
Listenfenster *330, 331*
Botschafts-Konstanten *75*
Erzeugen *323, 324, 325, 326, 327, 328, 329*
LmParent Konstante *76*
Load Elementfunktion
TOcPart *734*
TOcRemView *747*
LoadAccelerators
TModule Elementfunktion *371*
LoadAcceleratorTable
TWindow Elementfunktion *572*
LoadBitmap
TModule Elementfunktion *371*
LoadBOle Elementfunktion
TOcRegistrar *744*
LoadCursor
TModule Elementfunktion *371*
LoadData
TEditView Elementfunktion *261*
TListView Elementfunktion *335*
LoadFile
TDib Elementfunktion *226*
LoadIcon
TModule Elementfunktion *371*
LoadMenu
TModule Elementfunktion *371*
LoadResource
TDib Elementfunktion *226*
TModule Elementfunktion *371*
LoadString
TModule Elementfunktion *372*
Locate Elementfunktion
TOcPartCollection *739*
Lock
TPrintDialog::TData Elementfunktion *437*
LockBuffer
TEdit Elementfunktion *249*
LockWindowUpdate
TWindow Elementfunktion *553*
LogFont
TChooseFontDialog::TData Datenelement *126*
Lokalisieren
String-Ressourcen *336, 337*
long far*() Elementfunktion
TAutoVal *691*
long() Elementfunktion
TAutoVal *691*
Lookup
TLookupValidator Elementfunktion *339*
TStringLookupValidator Elementfunktion *496*
Lookup Elementfunktion
TAutoProxy *679*
LookupError Elementfunktion
TAutoCommand *662*
LowerBound

TSortedStringArray
Elementfunktion *483*
LowMemory
TModule Elementfunktion *372*
LowValue
TScrollBar Datenelement *467*
LParam
Parameter einer Dialogelementbotschaft *217*
lpCmdLine
TModule Datenelement *367*
LPtoDP
TDC Elementfunktion *184*
LPtoSDP
TPrintPreviewDC Elementfunktion *446*
LtBlue
TColor Datenelement *135*
LtCyan
TColor Datenelement *135*
LtGray
TColor Datenelement *135*
LtGreen
TColor Datenelement *135*
LtMagenta
TColor Datenelement *135*
LtRed
TColor Datenelement *136*
LtYellow
TColor Datenelement *136*

M

MACROGEN Dienstprogramm *617*
Magnitude
TSize Elementfunktion *817*
MainWindow
Variable *92*
MakeWindow
TModule Elementfunktion *372*
Makro _ICLASS *613*
Makro _IFUNC *613*
Makros
Anwendungs-Dictionary *60*
Automatisierungs-Server *604ff*
Automatisierungskontroller *607, 616, 617, 624*
COM-Objekte erzeugen *634*
DEFINE_RESPONSE_TABLE *60*
Deklarationsbezeichner *613*
Ereignisbehandlung *59, 64*
Hook-Prozeduren *606*
Klassenmodifizierer *618*
REQUIRED_ARG *659*
TButton Klasse *108*
TChooseColorDialog Klasse *123*
TChooseFontDialog Klasse *125*
TClipboardViewer Klasse *134*
TDecoratedFrame Klasse *211*
TDocManager Klasse *242*
TFloatingFrame Klasse *276*
TGdiObject Klasse *306*
Makros _OCFxxxx *613*
Map
TDib Typdefinition *219*
MapColor
TDib Elementfunktion *223*
MapIndex
TDib Elementfunktion *223*
MappingMode
TMetaFilePict Elementfunktion *365*
MapStatusCodeToString
TXCompatibility Öffentliche Elementfunktion *577*
TXOutOfMemory Öffentliche Elementfunktion *578*
MapUIColors
TDib Elementfunktion *223*
MapWindowPoints
TWindow Elementfunktion *553*
Margin
TGauge Datenelement *301*
Margins
TGadget Datenelement *289*
TGadgetWindow Datenelement *297*
Markierungsfelder
Zustand *119*
Maße
Layout *76*
MatchTemplate
TDocManager Elementfunktion *240*
Matrix
Toolbox-Anordung *506*
Max
TGauge Datenelement *301*
TRangeValidator Datenelement *455*
TSlider Elementfunktion *477*
MAX_RSRC_ERROR_STRING
Konstante *76*
Maximum-Werte

suchen *455*
MaxWidth
TListView Datenelement *333*
MB_Xxxx
Botschafts-Konstanten *76*
MB_Xxxx Konstanten und
Dokumentverwaltung *240*
MB_Xxxx Konstanten und TDocument *232*
MDIFILE.CPP
Illustration von Exceptions *579*
MeasureItem
TControl Elementfunktion *158*
TMenu Elementfunktion *353*
Mehrbenutzerbetrieb *781*
Mem Datenelement
TOleAllocator *766*
Menu
TWindowAttr-Struktur *529*
Menüdeskriptor
Löschen *279*
MenuItemId
TDecoratedFrame Datenelement *210*
menuname Registrierschlüssel *649*
Menüobjekte *349, 350, 351, 352, 353, 354*
Menüressource-ID *529*
Menüs
erstellen *496*
ID-Konstanten *67*
Systemmenüs *496*
Merge
TMenuDescr Elementfunktion *360*
MergeMenu
TFrameWindow Elementfunktion *280*
MergeModule
TFrameWindow Datenelement *282*
MessageBox
TWindow Elementfunktion *554*
MessageLoop
TApplication Elementfunktion *88*
MessageLoopResult
TApplication Datenelement *90*
Method
TVbxControl Elementfunktion *518*
Min
TGauge Datenelement *301*
TRangeValidator Datenelement *455*
TSlider Elementfunktion *477*
Mm
TMetaFilePict Datenelement *366*
Modale Dialogfenster *212*
Modales Dialogfenster *213*
Mode
TDib Datenelement *225*
ModeIndicators
TStatusBar Datenelement *489*
ModeIndicatorState
TStatusBar Datenelement *489*
ModifyMenu
TMenu Elementfunktion *353*
ModifyWorldTransform
TDC Elementfunktion *186*
Module *83*
TMenuDescr Datenelement *360*
TModule Datenelement *367*
Modus-Konstanten *67*
Modusanzeigen *487*
Rahmen *489*
MostDerived Funktion *650*
MouseEnter
TButtonGadget Elementfunktion *113*
TGadget Elementfunktion *290*
MouseLeave
TButtonGadget Elementfunktion *113*
TGadget Elementfunktion *290*
MouseMove
TButtonGadget Elementfunktion *113*
TGadget Elementfunktion *290*
MouseOffset
TSlider Elementfunktion *477*
Move
TUIHandle Elementfunktion *510*
TVbxControl Elementfunktion *518*
MoveTo
TDC Elementfunktion *187*
TUIHandle Elementfunktion *510*
MoveWindow
TWindow Elementfunktion *554*
Msg
TEventInfo Datenelement *263*
TGdiObject::TXGdi Elementfunktion *308*
TResponseTableEntry Datenelement *460*
TXWindow Elementfunktion *576*
Multidokumentschnittstelle
Client-Fenster *341ff*
Hauptfenster *345, 346, 347*
MustBeBound Elementfunktion
TAutoProxy *680*
MyEdge
TLayoutConstraint Datenelement *316*

N

Name
TDialogAttr Datenelement *219*
TModule Datenelement *373*
NBits *77*
NCels
TCelArray Datenelement *117*
nCmdShow
TApplication Datenelement *84*
NColors *78*
NewFile
TEditFile Elementfunktion *256*
NewStringList
TStringLookupValidator Elementfunktion *496*
Next
TDocument::List Elementfunktion *235*
TWindow Elementfunktion *554*
NextBand
TPrintDC Elementfunktion *432*
NextGadget
TGadget Elementfunktion *287*
TGadgetWindow Elementfunktion *295*
NextStream
TDocument Elementfunktion *231*
TStream Datenelement *495*
NextView
TDocument Elementfunktion *231*
Nicht-modales Dialogfenster
erzeugen *460*
nitInfo Elementfunktion
TOcDragDrop *712*
Non-Client-Ereignisse *499, 500, 501, 502, 503, 504, 505*
Normalize
TRect Elementfunktion *810*
Normalized
TRect Elementfunktion *810*
NotchCorners
TButtonGadget Datenelement *111*
NotifyCodeTResponseTableEntry
Datenelement *460*
NotifyParent
TGroupBox Datenelement *309*
TSlider Elementfunktion *476*
TVSlider Elementfunktion *480*
NotifyViews
TDocument Elementfunktion *231*
NotOK
TView Datenelement *528*
NumCels
TCelArray Elementfunktion *116*
NumChars
TTextGadget Elementfunktion *498*
NumChildren
TWindow Elementfunktion *555*
NumClrs
TDib Datenelement *225*
numColors
TDib Elementfunktion *223*
NumColumns
TToolBox Datenelement *506*
Numerische Werte
Maximum/Minimum setzen *455*
prüfen *454*
NumGadgets
TGadgetWindow Datenelement *297*
NumModeIndicators
TStatusBar Datenelement *489*
NumRowsTToolBox Datenelement *507*
NumScans
TDib Elementfunktion *223*

O

Oberflächenelemente
Autocreation deaktivieren *538*
automatisch erstellen *539*
Dialogelemente *156*
Dialogfenster *214*
Listenfenster *323ff*
löschen *538*
Oberflächenobjekte
anzeigen *565*
bedingtes Schließen *534*
Bildlauf *570*
Datentransfer *566*
Einrichtung *572*
Einstellen *284*
Fensterobjekte *570*
Flags setzen *561*
Registrierklassenname *570*
Statischer Text *484, 485*
Status *486, 532*
Transfermechanismus aktivieren *539*
Transfermechanismus deaktivieren *539*
Übertragung *566*
untergeordnete Fenster *536*

Object
TEventInfo Datenelement *263*
ObjectComponents
Aufzählungstypen verknüpfen und einbetten *611*
Automatisierungsklassen *602*
Bibliotheken *599*
Botschaften verknüpfen und einbetten *611*
Ereignisbotschaften *650, 651*
Exception-Klassen *601*
globale Funktionen *601*
Header-Dateien *600*
Klassen verknüpfen und einbetten *610*
Klassenmodifizierer *618*
Registrierungs-Strings lokalisieren *649*
Strukturen verknüpfen und einbetten *612*
ObjectComponents-Aufzählungstypen einbetten *611*
ObjectComponents-Aufzählungstypen verknüpfen *611*
ObjectComponents-Botschaften einbetten *611*
ObjectComponents-Botschaften verknüpfen *611*
ObjectComponents-Klassen einbetten *610*
ObjectComponents-Klassen verknüpfen *610*
ObjectComponents-Strukturen einbetten *612*
ObjectComponents-Strukturen verknüpfen *612*
ObjectPtr Typdefinition *650*
ObjectWindows *33ff*
Klassenhierarchie *33ff*
Überblick *33ff*
Objekte
Fenster *570*
Objekte automatisieren *604, 605*
Objekte im Dokument
Darstellung *704*
Objektgenerierung
Factory Klassen-Templates *73*
OC_APPBORDERSPACEREQ
Ereignisbehandlungsroutine *651*
OC_APPBORDERSPACESET
Ereignisbehandlungsroutine *651*
OC_APPDIALOGHELP
Ereignisbehandlungsroutine *651*
OC_APPFRAMERECT
Ereignisbehandlungsroutine *651*
OC_APPINSMENUS
Ereignisbehandlungsroutine *651*
OC_APPMENUS
Ereignisbehandlungsroutine *651*
OC_APPPROCESSMSG
Ereignisbehandlungsroutine *651*
OC_APPRESTOREUI
Ereignisbehandlungsroutine *651*
OC_APPSHUTDOWN
Ereignisbehandlungsroutine *651*
OC_APPSTATUSTEXT
Ereignisbehandlungsroutine *651*
OC_VIEWATTACHWINDOW
Ereignisbotschaft *652*
OC_VIEWBORDERSPACEREQ
Ereignisbotschaft *652*
OC_VIEWBORDERSPACESET
Ereignisbotschaft *652*
OC_VIEWCLIPDATA Ereignisbotschaft *652*
OC_VIEWCLOSE Ereignisbotschaft *652*
OC_VIEWDRAG Ereignisbotschaft *652*
OC_VIEWDROP Ereignisbotschaft *652*
OC_VIEWGETPALETTE Ereignisbotschaft *652*
OC_VIEWGETSCALE Ereignisbotschaft *652*
OC_VIEWGETSITERECT
Ereignisbotschaft *652*
OC_VIEWINSMENUS Ereignisbotschaft *652*
OC_VIEWLOADPART Ereignisbotschaft *652*
OC_VIEWOPENDOC Ereignisbotschaft *652*
OC_VIEWPAINT Ereignisbotschaft *652*
OC_VIEWPARTINVALID
Ereignisbotschaft *652*
OC_VIEWPARTSIZE Ereignisbotschaft *652*
OC_VIEWSAVEPART Ereignisbotschaft *652*
OC_VIEWSCROLL Ereignisbotschaft *652*
OC_VIEWSETSCALE Ereignisbotschaft *652*
OC_VIEWSETSITERECT Ereignisbotschaft *652*
OC_VIEWSHOWTOOLS Ereignisbotschaft *652*
OC_VIEWTITLE Ereignisbotschaft *652*
OcApp

TOleFrame Datenelement, geschütztes *385*
OcDocument Datenelement
TOcView *763*
ocreg.h *612*
ocrxxxx-Konstanten *612*
Aktionsart-Attribut- *656*
Aktionsartmenü-Flags *656*
Begrenzungs- *654*
Medien- *654*
Objektstatus- *655*
Richtungs- *654*
Verwendungs- *656*
Zwischenablage *653*
ocrxxxxKonstanten
Aspekte *652*
ODADrawEntire
TControl Elementfunktion *158*
ODAFocus
TControl Elementfunktion *158*
ODASelect
TControl Elementfunktion *158*
Oerflächenobjekte
Einstellen *211*
Öffentliche Datenelemente
TApplication Klasse *83*
TButton Klasse *107*
TCheckBox Klasse *118*
TChooseColorDialog Klasse *122*
TChooseColorDialog::TData Struktur *123*
TChooseFontDialog::TData Struktur *125*
TClipboard Klasse *129*
TComboBox Klasse *139*
TCommandEnabler Klasse *150*
TDialog Klasse *213*
TDocManager Klasse *237*
TDocument Klasse *227*
TEditFile Klasse *255*
TEditSearch Klasse *258*
TEventInfo Klasse *263*
TFindReplaceDialog::TData Klasse *273*
TFrameWindow Klasse *279*
TGroupBox Klasse *309*
THatch8x8Brush Klasse *311*
TInputDialog Klasse *314*
TLayoutConstraint Struktur *316*
TLayoutMetrics Klasse *318*
TListView Klasse *332*
TMDIClient Klasse *342*
TModule Klasse *367*
TPaintDC Klasse *414*
TPen Klasse *419*
TPrintDialog::TData Struktur *435*
TResponseTableEntry Klasse *460*
TScrollBar Klasse *463*
TStatic Klasse *484*
TView Klasse *526*
TWindow Klasse *531*
TWindowAttr-Struktur *529*
Öffentliche Elementfunktionen *806*
TApplication Klasse *85*
TBitmapGadget Klasse *102*
TBitSet Klasse *103*
TCelArray Klasse *116*
TCharSet Klasse *118*
TCheckBox Klasse *119*
TChooseColorDialog Klasse *122*
TClipboard Klasse *129*
TComboBox Klasse *140*
TComboBoxData Klasse *144*
TCommandEnabler Klasse *150*
TCommonDialog Klasse *152*
TControl Klasse *157*
TDecoratedFrame Klasse *209*
TDialog Klasse *214*
TDocManager Klasse *237*
TDocTemplate Klasse *795*
TDocTemplateT Klasse *800*
TDocument Klasse *228*
TEdgeConstraint Struktur *243*
TEdgeOrSizeConstraint Struktur *244*
TEdit Klasse *246*
TEditFile Klasse *255*
TEditSearch Klasse *258*
TEditView Klasse *260*
TEventHandler Klasse *264*
TFileDocument Klasse *266*
TFileOpenDialog Klasse *269*
TFileSaveDialog Klasse *269*
TFilterValidator Klasse *270*
TFindReplaceDialog Klasse *272*
TFloatingFrame Klasse *276*
TFrameWindow Klasse *279*
TGadgetWindow Klasse *293*
TGauge Klasse *300*
TGdiObject Klasse *304*
TGroupBox Klasse *310*
TIcon Klasse *313*
TInputDialog Klasse *315*
TLayoutWindow Klasse *322*
TListBox Klasse *324*

TListView Klasse *332*
TLocaleString Struktur *337*
TLookupValidator Klasse *338*
TMDIChild Klasse *340*
TMDIClient Klasse *342*
TMDIFrame Klasse *346*
TMemoryDC Klasse *348*
TMenuDescr Klasse *359*
TMenuItemEnabler Klasse *361*
TMessageBar Klasse *362*
TMetaFileDC Klasse *364*
TModule Klasse *368*
TModule::TXInvalidModule Klasse *374*
TOpenSaveDialog *413*
TOpenSaveDialog Klasse *409*
TPalette Klasse *415*
TPen Klasse *420*
TPopupMenu Klasse *422*
TPreviewPage Klasse *423*
TPrintDC Klasse *424*
TPrintDialog Klasse *438*
TPrintDialog::TData Struktur *436*
TPrinter Klasse *439*
TPrinterAbortDlg Klasse *442*
TPrintout Klasse *443*
TPrintPreviewDC Klasse *446*
TProfile Klasse *450*
TPXPictureValidator Klasse *451*
TRangeValidator Klasse *454*
TScroller Klasse *469*
TSlider Klasse *473*
TStringLookupValidator Klasse *496*
TValidator Klasse *511*
TVbxControl Klasse *515*
TView Klasse *526*
TWindow Klasse *533*
TWindowView Klasse *575*
Öffentliche Elementfunktionen, TMenu Klasse *350*
Öffentliche Elementfunktionen, TMetaFilePict Klasse *365*
Öffentliche Konstruktoren
TFloatingFrame Klasse *275*
Öffnen einer Datei
Dateifehler *268*
Offs
TCelArray Datenelement *117*
Offset
TCelArray Elementfunktion *116*
TRect Elementfunktion *811*
OffsetBy
TRect Elementfunktion *811*
OffsetClipRgn
TDC Elementfunktion *187*
OffsetViewportOrg
TDC Elementfunktion *187*
TPrintPreviewDC Elementfunktion *446*
OffsetWindowOrg
TDC Elementfunktion *187*
ofn
TOpenSaveDialog Datenelement *409*
ofxxxx Dokument-Öffnen-Aufzählung (Konstanten) *78*
OLE-Anwendungen *600*
Argumente Registrieren *631*
Ausführungsmodi *705*
Automatisierungsbefehle *607*
Ereignisse *650*, *651*
Schnittstellen, Implemetierung *613*
Zugreifen auf automatisierte Klassen *604*, *605*
Zwischenablageformate *646*
OLE-Funktionen
modifizieren *613*
OLE-Programme
Konstanten *78*
OLE-Tastenkürzel
Ressourcen-IDs *66*
Open
TDocument Elementfunktion *231*
TEditFile Elementfunktion *256*
TFileDocument Elementfunktion *267*
TOleDocument Elementfunktion, öffentliche *376*
TStorageDocument Elementfunktion *492*
Open Elementfunktion
TOcPart *734*
OpenClipboard
TClipboard Elementfunktion *131*
TWindow Elementfunktion *555*
OpenHandle
TStorageDocument Elementfunktion *492*
OpenThisFile
TFileDocument Elementfunktion *268*
operator!=
TRect Elementfunktion *812*
TRegion Elementfunktion *457*
TSize Elementfunktion *817*
operator *815*, *818*
Elementfunktion *683*

TStatus Datenelement *486*
Operator *813, 815*
Elementfunktion *727*
TCelArray Elementfunktion *116*
TSortedStringArray Elementfunktion *483*
Operator &=
TBitSet Elementfunktion *104*
Operator * *806*
operator ++ Elementfunktion
TOcPartCollectionIter *740*
Operator +=
TBitSet Elementfunktion *104*
Operator - *806*
Operator -=
TBitSet Elementfunktion *104*
operator <
TMetaFilePict Elementfunktion *365*
Operator < *813*
Operator = *806*
TCelArray Elementfunktion *116*
TLocaleString Elementfunktion *338*
TMenu Elementfunktion *353*
operator = Elementfunktion
TAutoString *685*
TAutoVal *688*
TOcScaleFactor *750*
Operator ==
TDib Elementfunktion *223*
TModule Elementfunktion *372*
operator == Elementfunktion
TOcPart *729*
Operator |=
TBitSet Elementfunktion *104*
Operator ~
TBitSet Elementfunktion *104*
Operator BITMAPINFO()
TDib Elementfunktion *221*
Operator BITMAPINFOHEADER()
TDib Elementfunktion *221*
Operator COLORREF()
TColor Elementfunktion *138*
operator const char*
TLocaleString Elementfunktion *337*
operator double Elementfunktion
TAutoDate *666*
Operator HANDLE()
TDib Elementfunktion *222*
Operator HBITMAP()
TBitmap Elementfunktion *101*
operator HDC()
TDC Elementfunktion *183*
Operator HDROP()
TDropInfo Elementfunktion *802*
Operator HICON
TIcon Elementfunktion *314*
operator HINSTANCE
TModule Elementfunktion *372*
operator HMENU
TMenu Elementfunktion *353*
operator HMETAFILE()
TMetaFilePict Elementfunktion *365*
operator HPALETTE()
TPalette Elementfunktion *416*
operator HPEN()
TPen Elementfunktion *421*
operator HRGN()
TRegion Elementfunktion *458*
operator int Elementfunktion
TOcPartCollectionIter *740*
operator IUnknown&() Elementfunktion
TUnknown *774*
operator IUnknown*() Elementfunktion
TUnknown *774*
Operator TBitmap&
TCelArray Elementfunktion *116*
Operator TPoint*
TRect Elementfunktion *813*
Operator TRgbQuad()
TDib Elementfunktion *224*
operator uint
TMenu Elementfunktion *354*
Operator&
TRect Elementfunktion *812*
operator&=
TRect Elementfunktion *812*
TRegion Elementfunktion *458*
Operator+
TRect Elementfunktion *811*
operator+=
TRect Elementfunktion *812*
TRegion Elementfunktion *457*
TSize Elementfunktion *817*
Operator-
TRect Elementfunktion *811*
TSize Elementfunktion *817*
operator-=
TRect Elementfunktion *812*
TRegion Elementfunktion *458*
TSize Elementfunktion *817*
operator<

TPalette Elementfunktion *416*
TResID Friend *815*
TSize-Friend *818*
operator=
TRegion Elementfunktion *457*
TStatus Datenelement *486*
Operator==
TColor Elementfunktion *137*
TRect Elementfunktion *812*
TRegion Elementfunktion *457*
TSize Elementfunktion *817*
operator^=
TRegion Elementfunktion *458*
Operator|
TRect Elementfunktion *812*
operator|=
TRect Elementfunktion *813*
TRegion Elementfunktion *458*
Operatoren *806, 815*
TStatusBar Klasse *488*
OPTIONAL_ARG Makro *657*
Options
TValidator Datenelement *513*
OrgBitmap
TMemoryDC Datenelement *349*
OrgBrush
TDC Datenelement *207*
OrgFont
TDC Datenelement *207*
OrgPalette
TDC Datenelement *207*
OrgPen
TDC Datenelement *208*
OrgTextBrush
TDC Datenelement *208*
Origin
TEditView Datenelement *261*
TListView Datenelement *333*
Origin Datenelement
TOcView *763*
OtherEdge
TLayoutConstraint Datenelement *316*
Outer Datenelement
TUnknown *775*
OutStream
TDocument Elementfunktion *232*
TFileDocument Elementfunktion *267*
TStorageDocument Elementfunktion *493*
OWL\INCLUDE Verzeichnis
und Header-Dateien *50*
OWLDialog
ObjectWindows Standardklasse *217*
OWLGetVersion *82*
Owner Datenelement
TAutoIterator *673*
TAutoStack *683*

P

PageMagnitude
TScrollBar Datenelement *463*
PageNum
TPreviewPage Datenelement *423*
PageSize
TPrintout Datenelement *445*
Paint
TBitmapGadget Elementfunktion *103*
TButtonGadget Elementfunktion *114*
TGadget Elementfunktion *290*
TGadgetWindow Elementfunktion *299*
TGauge Elementfunktion *301*
TOleWindow Elementfunktion, geschützte *406*
TPreviewPage Elementfunktion *423*
TTextGadget Elementfunktion *498*
TUIHandle Elementfunktion *510*
TWindow Elementfunktion *555*
PAINT.CPP Beispielprogramm *275*
PaintBorder
TGadget Elementfunktion *290*
TGauge Elementfunktion *302*
PaintButton
TTinyCaptiojn Elementfunktion *505*
PaintCaption
TTinyCaption Elementfunktion *505*
PaintCloseBox
TTinyCaption Elementfunktion *505*
PaintGadgets
TGadgetWindow Elementfunktion *299*
TMessagebar Elementfunktion *363*
PaintInfo-Struktur *555*
PaintMaxBox
TTinyCaption Elementfunktion *505*
PaintMetafile
TOleWindow Datenelement, geschützes *394, 395, 396*
TOleWindow Elementfunktion, öffentliche *394*
PaintMinBox

TTinyCaption Elementfunktion *505*
PaintParts
TOleWindow Elementfunktion, geschützte *406*
PaintRgn
TDC Elementfunktion *187*
PaintRuler
THSlider Elementfunktion *479*
TSlider Elementfunktion *476*
TVSlider Elementfunktion *480*
PaintSlot
THSlider Elementfunktion *479*
TSlider Elementfunktion *476*
TVSlider Elementfunktion *481*
PaintSysBoxTinyCaption Elementfunktion *505*
PaintThumb
TSlider Elementfunktion *476*
PalIndex
TColor Elementfunktion *138*
PalRelative
TColor Elementfunktion *138*
Param
TDialogAttr Datenelement *219*
TWindowAttr-Struktur *529*
Parameter
automatisierte Methoden *660*
Parametrisierte Subklasse
View- und Dokumentklassen *799*
Parent
TWindow Datenelement *532*
Part Datenelement
TOcViewPaint *764*
PartSize Datenelement
TOcScaleFactor *750*
Paste
TEdit Elementfunktion *250*
Paste Elementfunktion
TOcApp *700*
TOcView *759*
Paßwörter *248*, *250*
PatBlt
TDC Elementfunktion *188*
path Registrierschlüssel *657*
PathChanged
TOleDocument Elementfunktion, öffentliche *377*
PathToRegion
TDC Elementfunktion *188*
pd
TPrintDialog Datenelement *438*
PercentOf
TEdgeConstraint Elementfunktion *243*
TEdgeOrSizeContraint Elementfunktion *244*
PerformCreate
TEditView Elementfunktion *260*
TMDIFrame Elementfunktion *347*
TVbxControl Elementfunktion *520*
TWindow Elementfunktion *555*
permid Registrierschlüssel *658*
permname Registrierschlüssel *658*
pfxxxx Eigenschaftsattributkonstanten *79*
Pic
TPXPictureValidator Datenelement *452*
Picture
TPXPictureValidator Elementfunktion *452*
Pie
TDC Elementfunktion *188*
Planes
TBitmap Elementfunktion *101*
PlayMetaFile
TDC Elementfunktion *189*
PlayMetaFileRecord
TDC Elementfunktion *189*
PlayOnto
TMetaFilePict Elementfunktion *365*
PlgBlt
TDC Elementfunktion *189*
Pmf
TResponseTableEntry Datenelement *460*
PointSize
TChooseFontDialog::TData Datenelement *126*
PointToPos
THSlider Elementfunktion *479*
TSlider Elementfunktion *476*
TVSlider Elementfunktion *481*
PolyBezier
TDC Elementfunktion *190*
PolyBezierTo
TDC Elementfunktion *190*
PolyDraw
TDC Elementfunktion *191*
Polygon
TDC Elementfunktion *192*
Polyline
TDC Elementfunktion *192*
PolylineTo
TDC Elementfunktion *192*

PolyPolygon
TDC Elementfunktion *192*
PolyPolyline
TDC Elementfunktion *193*
Pos
TSlider Datenelement *477*
Pos Datenelement
TOcViewPaint *764*
Position
Cursor, zurückgeben *250*
aktuelle *249, 250, 565*
Angegeben durch Variablen *247*
Cursor, zurückgeben *248*
durch Variablen angegeben *251*
Editierfelder *251*
im Verhältnis zum Ursprung *453*
relativ zum Ursprung *107, 139, 157, 246*
TMenuItemEnabler Datenelement *361*
TScrollBar Datenelement *467*
Verhältnis zum Ursprung *484*
Zeichen, Editierfeld *245, 248*
PositionGadget
TControlBar Elementfunktion *154*
TGadgetWindow Elementfunktion *299*
TStatusBar Elementfunktion *489*
Positionszeiger
Position *471*
PostDispatchAction
TApplication Elementfunktion *88*
PostDocError
TDocManager Elementfunktion *240*
PostError
TDocument Elementfunktion *232*
PostMessage
TWindow Elementfunktion *555*
PosToPoint
THSlider Elementfunktion *480*
TSlider Elementfunktion *476*
TVSlider Elementfunktion *481*
PreOpen
TOleDocument Elementfunktion, öffentliche *377*
PreProcessMenu
TApplication Elementfunktion *88*
PreProcessMsg
TControlBar Elementfunktion *154*
TDecoratedFrame Elementfunktion *210*
TDialog Elementfunktion *217*
TFrameWindow Elementfunktion *280*
TMDIClient Elementfunktion *343*
TStatusBar Elementfunktion *490*
TWindow Elementfunktion *555*
Pressed
TButtonGadget Datenelement *111*
Previous
TWindow Elementfunktion *556*
Print
TPrinter Elementfunktion *440*
PrintDC
TPreviewPage Datenelement *423*
PrintExtent
TPreviewPage Datenelement *423*
Printout
TPreviewPage Datenelement *423*
PrintPage
TPrintout Elementfunktion *444*
PrnDC
TPrintPreviewDC Datenelement *449*
PrnFont
TPrintPreviewDC Datenelement *449*
ProcessAppMsg
TApplication Elementfunktion *89*
ProcessCmdLine Elementfunktion
TRegistrar *770*
progid Registrierschlüssel *658*
Programme
Statusanzeige *486, 489*
prompt
TInputDialog Datenelement *314*
Property enum
TView Datenelement *526*
PropertyCount
TDocument Elementfunktion *232*
TStorageDocument Elementfunktion *493*
TView Elementfunktion *527*
PropertyFlags
TDocument Elementfunktion *232*
TFileDocument Elementfunktion *267*
TStorageDocument Elementfunktion *493*
TView Elementfunktion *527*
PropertyName
TDocument Elementfunktion *232*
TFileDocument Elementfunktion *267*
TStorageDocument Elementfunktion *493*
TView Elementfunktion *527*
Prüfen von Editierobjekten und abgeleiteten Klassen *252*
Prüfung abstrakter Daten *511, 512, 513*
Prüfung der Benutzereingabe *270*
Prüfungsobjekte

Bereiche *454*
Datentransfer *455*
Filter *270*
Gültigkeitsprüfung *270*
IDs *70*
Ps
TPaintDC Datenelement *414*
PtIn
TGadget Elementfunktion *291*
PtVisible
TDC Elementfunktion *193*
Puffer
Datentransfer *466, 484, 485, 507, 566*
Datenübertragung *563, 566*
Datenübertragung aktivieren *539*
Datenübertragung in *507*
Ermitteln von Handles *247, 250*
Text *246, 247*
Transfer, Kombinationsfenster *143, 144*
Zeiger auf Transferdaten *569*
PumpWaitingMessages
TApplication Elementfunktion *89*

Q

QueryAbort
TPrintDC Elementfunktion *432*
QueryCreate
TClipboard Elementfunktion *132*
QueryEscSupport
TPrintDC Elementfunktion *433*
QueryLink
TClipboard Elementfunktion *132*
QueryObject Elementfunktion
TUnknown *775*
QueryThrow
TApplication Elementfunktion *89*
QueryViews
TDocument Elementfunktion *232*

R

Range
TSlider Datenelement *477*
rdSetData-Konstante *466*
Read
TDib Elementfunktion *226*
TEditFile Elementfunktion *256*
TOleDocument Elementfunktion, öffentliche *377*
RealizePalette
TDC Elementfunktion *193*
ReceiveMessage
TWindow Elementfunktion *556*
Rechteck aktualisieren
Fenster *548*
Reconstruct
THatch8x8Brush Elementfunktion *312*
Record Elementfunktion
TAutoCommand *663*
Rectangle
TDC Elementfunktion *193*
RectVisible
TDC Elementfunktion *193*
Red
TColor Elementfunktion *138*
RedrawWindow
TWindow Elementfunktion *556*
RefAdd
TGdiObject Elementfunktion *304*
RefCount
TGdiObject Elementfunktion *304*
RefDec
TGdiObject Elementfunktion *304*
RefFind
TGdiObject Elementfunktion *304*
RefInc
TGdiObject Elementfunktion *305*
RefRemove
TGdiObject Elementfunktion *305*
Refresh
TVbxControl Elementfunktion *518*
RefTemplate
TDocManager Elementfunktion *241*
Register
TWindow Elementfunktion *557*
RegisterAppClass Elementfunktion
TRegistrar *770*
RegisterClass Elementfunktion
TOcApp *701*
RegisterClasses Elementfunktion
TOcApp *701*
RegisterClipboardFormat
TClipboard Elementfunktion *132*
RegisterClipFormats Elementfunktion
TOcView *759*
RegisterHotKey
TWindow Elementfunktion *557*

RegisterWindowMessage
TEventHandler Elementfunktion *264*
Registrar-Objekt *741*
Registrieren von
ObjectComponents-Anwendungen *608*
Registriermakro
END_REGISTRATION_makro *789*
IREGFORMAT_makro *791*
IREGICON_makro *793*
IREGSTATUS_makro *792*
REGDOCFLAGS_makro *794*
REGISTRATION_FORMAT_BUFFER_makro *794*
REGVERBOPT_makro *793*
Registriermakros *787, 789*
IREGDATA_makro *789*
Registrierschlüssel *608*
aspectall *614*
aspectcontent *614*
aspectdocprint *615*
aspecticon *615*
aspectthumbnail *616*
Dateinamenerweiterung *645*
description *636*
directory *637*
Dokumentfilter *637*
global eindeutige Bezeichner *630*
Hilfedateien *647*
Locale-ID *649*
OLE-Anwendungen *631*
Server *649*
Server Debuggen *631*
Server-Namen *649*
Standard-Dateiformate *646*
Symbole *648*
typehelp *781*
usage *781*
verbn *782*
verbnopt *782*
version *783*
Verzeichnispfad *647, 657*
Zwischenablage *646*
Registriertabellen *630*
konstruieren *612*
Registrierung
Fenster *557*
Registrierungsdatenbanken
Einträge vornehmen *608*
Registrierungsmakro
REGITEM_makro *791*
Relationship
TLayoutConstraint Datenelement *316*
Release Elementfunktion
TOcSaveLoad *748*
ReleaseAutoApp Elementfunktion
TRegistrar *770*
ReleaseCapture
TWindow Elementfunktion *557*
ReleaseDoc
TOleDocument Elementfunktion, öffentliche *377*
TStorageDocument Elementfunktion *493*
ReleaseGlyphDib
TButtonGadget Elementfunktion *114*
ReleaseObject Elementfunktion
TOcApp *701*
TOcView *760*
RelWin
TLayoutConstraint Datenelement *316*
Remove
TAppDictionary Elementfunktion *96*
TDocument::List Elementfunktion *235*
TGadgetWindow Elementfunktion *295*
RemoveChild
TWindow Elementfunktion *572*
RemoveChildLayoutMetrics
TLayoutWindow Elementfunktion *322*
Removed
TControlGadget Elementfunktion *156*
TGadget Elementfunktion *291*
RemoveItem
TVbxControl Elementfunktion *518*
RemoveMenu
TMenu Elementfunktion *354*
RemoveProp
TWindow Elementfunktion *557*
Rename Elementfunktion
TOcPart *734*
TOcRemView *747*
TOcView *760*
ReOrg
TPrintPreviewDC Elementfunktion *446*
Repeat
TButtonGadget Datenelement *111*
ReplaceWith
TEditFile Elementfunktion *256*
TFindReplaceDialog::TData Datenelement *274*
Report Elementfunktion
TAutoCommand *663*

ReportError
TPrinter Elementfunktion *440*
REQUIRED_ARG Makro *659*
ReScale
TPrintPreviewDC Elementfunktion *447*
ResetDC
TDC Elementfunktion *194*
ResetSelections
TComboBox Elementfunktion *145*
TListBoxData Elementfunktion *331*
ResId
TButtonGadget Datenelement *111*
ResizePalette
TPalette Elementfunktion *417*
ResourceIdToString
TXOwl Elementfunktion *580*
Ressource-ID
TXOwl *580*
Ressourcen-IDs
Modus-Konstanten *67*
Vorgabewert ermitteln *216*
Ressourcendateien
beschrieben *50*
Restart Elementfunktion
TOcPartCollectionIter *740*
RestoreBitmap
TMemoryDC Elementfunktion *348*
RestoreBrush
TDC Elementfunktion *194*
RestoreDC
TDC Elementfunktion *194*
RestoreFont
TDC Elementfunktion *194*
TPrintPreviewDC Elementfunktion *447*
RestoreMemory
TModule Elementfunktion *372*
RestoreMenu
TFrameWindow Elementfunktion *281*
RestoreObjects
TDC Elementfunktion *194*
TMemoryDC Elementfunktion *348*
RestorePalette
TDC Elementfunktion *194*
RestorePen
TDC Elementfunktion *194*
RestoreTextBrush
TDC Elementfunktion *194*
ResumeThrow
TApplication Elementfunktion *89*
Return Elementfunktion
TAutoCommand *663*
TAutoIterator *671*
Revert
TDocument Elementfunktion *233*
TFileDocument Elementfunktion *267*
TStorageDocument Elementfunktion *493*
Rgb
TColor Elementfunktion *138*
RightOf
TEdgeConstraint Elementfunktion *243*
RootDocument
TDocument Elementfunktion *233*
RoundRect
TDC Elementfunktion *195*
Rückgabecodes, OLE *648*
Run
TApplication Elementfunktion *89*
Run Elementfunktion
TRegistrar *771*

S

SameAs
TEdgeConstraint Elementfunktion *243*
TEdgeOrSizeContraint Elementfunktion *245*
Save
TEditFile Elementfunktion *257*
Save Elementfunktion
TOcPart *734*
TOcRemView *747*
SaveAs
TEditFile Elementfunktion *257*
SaveDC
TDC Elementfunktion *195*
SB_ Bildlaufleisten-Konstanten *471*
SB_Konstanten *470*
SB_Xxxx-Konstanten *82*
SBBottom
TScrollBar Elementfunktion *465*
SBBottom-Konstante *465*
SBLineDown
TScrollBar Elementfunktion *465*
SBLineDown-Konstante *465*
SBLineUp
TScrollBar Elementfunktion *465*
SBLineUp-Konstante *465*
SBPageDown-Konstante *465*
SBPageUp

TScrollBar Elementfunktion *465*
SBPageUp-Konstante *465*
SBS_HORZ-Konstante *463*
SBThumbPosition
TScrollBar Elementfunktion *465*
SBThumbPosition-Konstante *465*
SBThumbTrack
TScrollBar Elementfunktion *466*
SBThumbTrack-Konstante *466*
SBTop
TScrollBar Elementfunktion *466*
SBTop-Konstante *466*
ScaleViewportExt
TDC Elementfunktion *195*
TPrintPreviewDC Elementfunktion *447*
ScaleWindowExt
TDC Elementfunktion *195*
TPrintPreviewDC Elementfunktion *447*
Schalter-Flag-Konstanten *56*
Schalter-Gadgets der Dialogelementleiste *153*
Schalterbotschaftskonstanten *56*
Schaltereinstellungen *108*
Schalterstatus *56*
Schaltfelder *453*
Checkmarken *453*
Schließsymbole
erzeugen *499*
Schriften
Ändern *216*
Auswählen *124*
ScreenToClient
TWindow Elementfunktion *557*
Scroll
TEdit Elementfunktion *250*
ScrollBy
TScroller Elementfunktion *470*
ScrollDC
TDC Elementfunktion *196*
Scroller
TWindow Datenelement *532*
ScrollTo
TScroller Elementfunktion *470*
ScrollWindow
TWindow Elementfunktion *558*
ScrollWindow Elementfunktion
TOcView *760*
ScrollWindowEx
TWindow Elementfunktion *558*
SDPtoLP
TPrintPreviewDC Elementfunktion *447*
Search
TEdit Elementfunktion *250*
SearchCmd
TEditSearch Datenelement *258*
SearchData
TEditSearch Datenelement *258*
SearchDialog
TEditSearch Datenelement *258*
SelCount
TListBoxData Datenelement *330*
Select
TComboBox Elementfunktion *145*
TListBoxData Elementfunktion *331*
TOleWindow Elementfunktion, geschützte *406*
Select Elementfunktion
TOcPart *735*
SelectAll Elementfunktion
TOcPartCollection *739*
SelectAnySave
TDocManager Elementfunktion *241*
SelectClipPath
TDC Elementfunktion *196*
SelectClipRgn
TDC Elementfunktion *196*
SelectDocType
TDocManager Elementfunktion *241*
SelectEmbedded
TOleWindow Elementfunktion, geschützte *406*
SelectImage
TBitmapGadget Elementfunktion *102*
Selection
TComboBoxData Datenelement *145*
SelectionChanged
TGroupBox Elementfunktion *310*
SelectObject
TDC Elementfunktion *197*
TMemoryDC Elementfunktion *348*
TPrintPreviewDC Elementfunktion *448*
SelectSave
TDocManager Elementfunktion *241*
TDocTemplate Elementfunktion *798*
SelectStockObject
TDC Elementfunktion *197*
TPrintPreviewDC Elementfunktion *448*
SelectString
TComboBox Elementfunktion *145*
TListBoxData Elementfunktion *331*

SelectViewType
TDocManager Elementfunktion *242*
SelIndex
TComboBoxData Datenelement *146*
SelIndices
TListBoxData Datenelement *330*
SelStrings
TListBoxData Datenelement *330*
sEmpty Elementfunktion
TOcFormatList *715*
SendDlgItemMessage
TWindow Elementfunktion *558*
SendDlgItemMsg
TDialog Elementfunktion *217*
SendMessage
TWindow Elementfunktion *559*
SendNotification
TWindow Elementfunktion *559*
Server
Daten übergeben *654*
Debuggen *631*
Einbenutzerbetrieb *656*
Mehrbenutzerbetrieb *656*
Namen registrieren *649*
parallele Verwendung *656*
Registrierung *649*
Verbunddokumente und *707*
Set
TEdgeConstraint Elementfunktion *244*
TRect Elementfunktion *813*
SetAbortProc
TPrintDC Elementfunktion *433*
SetActive Elementfunktion
TOcPart *735*
SetActiveWindow
TWindow Elementfunktion *559*
SetAntialiasEdges
TButtonGadget Elementfunktion *110*
SetBitmapBits
TBitmap Elementfunktion *101*
SetBitmapDimension
TBitmap Elementfunktion *101*
SetBkColor
TDC Elementfunktion *197*
TPrintPreviewDC Elementfunktion *448*
SetBkgndColor
TWindow Elementfunktion *559*
SetBkMode
TDC Elementfunktion *197*
SetBorders
TGadget Elementfunktion *287*
SetBorderStyle
TGadget Elementfunktion *287*
SetBound
TGadget Elementfunktion *288*
SetBounds
TBitmapGadget Elementfunktion *103*
TButtonGadget Elementfunktion *114*
TControlGadget Elementfunktion *156*
SetBoundsRect
TDC Elementfunktion *198*
SetBrushOrg
TDC Elementfunktion *198*
SetButtonState
TButtonGadget Elementfunktion *110*
SetButtonType
TButtonGadget Elementfunktion *110*
SetCaption
TDialog Elementfunktion *217*
TWindow Elementfunktion *559*
SetCapture
TWindow Elementfunktion *559*
SetCaretBlinkTime
TWindow Elementfunktion *559*
SetCaretIndex
TListBox Elementfunktion *327*
SetCaretPos
TWindow Elementfunktion *559, 560*
SetCelSize
TCelArray Elementfunktion *116*
SetCheck
TButtonGadgetEnabler Elementfunktion *115*
TCheckBox Elementfunktion *119*
TCommandEnabler Elementfunktion *150*
TMenuItemEnabler Elementfunktion *362*
SetChildLayoutMetrics
TLayoutWindow Elementfunktion *322*
SetClassLong
TWindow Elementfunktion *560*
SetClassWord
TWindow Elementfunktion *560*
SetClientWindow
TDecoratedFrame Elementfunktion *210*
TFrameWindow Elementfunktion *281*
SetClipboardData
TClipboard Elementfunktion *132*
SetClipboardViewer
TClipboard Elementfunktion *133*
SetColor

TDib Elementfunktion *224*
TGauge Elementfunktion *300*
SetColumnWidth
TListBox Elementfunktion *327*
SetCommandHook Elementfunktion
TAutoCommand *663*
SetCopyCount
TPrintDC Elementfunktion *433*
SetCursor
TWindow Elementfunktion *560*
SetDefaultExt
TDocTemplate Elementfunktion *798*
SetDefaultId
TDialog Elementfunktion *217*
SetDevMode
TPrintDialog::TData Elementfunktion *437*
SetDevNames
TPrintDialog::TData Elementfunktion *437*
SetDIBits
TDC Elementfunktion *198*
SetDIBitsToDevice
TDC Elementfunktion *199*
SetDirection
TGadgetWindow Elementfunktion *295*
TToolox Datenelement *506*
SetDirectory
TDocTemplate Elementfunktion *798*
SetDlgItemInt
TWindow Elementfunktion *560*
SetDlgItemText
TWindow Elementfunktion *561*
SetDocManager
TApplication Elementfunktion *92*
TDocTemplate Elementfunktion *798*
TDocument Elementfunktion *233*
SetDocPath
TDocument Elementfunktion *233*
TStorageDocument Elementfunktion *494*
SetDocTitle
TEditView Elementfunktion *260*
TFrameWindow Elementfunktion *281*
TListView Elementfunktion *333*
TOleView Elementfunktion, öffentliche *389*
TView Elementfunktion *528*
TWindow Elementfunktion *561*
TWindowView Elementfunktion *575*
SetEditSel
TComboBox Elementfunktion *142*
SetEmbedded
TDocument Elementfunktion *233*
SetEmpty
TRect Elementfunktion *813*
SetEnabled
TGadget Elementfunktion *288*
SetErrorMesgHook Elementfunktion
TAutoCommand *663*
SetExtendedUI
TComboBox Elementfunktion *142*
SetExtent
TListView Elementfunktion *335*
SetFileFilter
TDocTemplate Elementfunktion *798*
SetFileName
TEditFile Elementfunktion *257*
SetFilter
TOpenSaveDialog::TData
Elementfunktion *413*
SetFlag
TDocTemplate Elementfunktion *798*
TWindow Elementfunktion *561*
SetFlag Elementfunktion
TAutoCommand *664*
SetFocus
TWindow Elementfunktion *561*
SetHandle
TEdit Elementfunktion *250*
SetHintCommand
TGadgetWindow Elementfunktion *295*
SetHintMode
TGadgetWindow Elementfunktion *296*
SetHintText
TMessagebar Elementfunktion *362*
SetHorizontalExtent
TListBox Elementfunktion *327*
SetHost Elementfunktion
TOcPart *735*
SetIcon
TFrameWindow Elementfunktion *281*
SetIndex
TDib Elementfunktion *224*
SetInstance
TModule Elementfunktion *372*
SetItemData
TComboBox Elementfunktion *142*
TListBox Elementfunktion *327*
SetItemHeight
TComboBox Elementfunktion *142*
TListBox Elementfunktion *328*
SetItemRect
TListBox Elementfunktion *328*

SetLang Elementfunktion
TAutoProxy *680*
SetLed
TGauge Elementfunktion *300*
SetLink Elementfunktion
TOcView *760*
SetMainWindow
TApplication Elementfunktion *92*
SetMapMode
TDC Elementfunktion *199*
TPrintPreviewDC Elementfunktion *448*
SetMapperFlags
TDC Elementfunktion *199*
SetMappingMode
TMetaFilePict Elementfunktion *366*
SetMargins
TFloatingFrame Elementfunktionen *276*
TGadget Elementfunktion *288*
TGadgetWindow Elementfunktion *296*
SetMenu
TFrameWindow Elementfunktion *281*
TMDIFrame Elementfunktion *347*
TWindow Elementfunktion *561*
SetMenuDescr
TFrameWindow Elementfunktion *281*
SetMenuItemBitmaps
TMenu Elementfunktion *354*
SetMiterLimit
TDC Elementfunktion *200*
SetModeIndicator
TStatusBar Elementfunktion *488*
SetModule
TDocTemplate Elementfunktion *798*
TMenuDescr Elementfunktion *359*
TWindow Elementfunktion *562*
SetName
TModule Elementfunktion *372*
SetNext
TWindow Elementfunktion *562*
SetNotchCorners
TButtonGadget Elementfunktion *110*
SetNull *814*
SetNull Elementfunktionen *814*
SetNumCels
TCelArray Elementfunktion *117*
SetOcApp
TOleFrame Elementfunktion, öffentliche *382*
SetOcDoc
TOleDocument Elementfunktion, öffentliche *377*
SetOffset
TCelArray Elementfunktion *116*
SetOpenMode
TDocument Elementfunktion *233*
SetOption
TValidator Elementfunktion *512*
SetOption Elementfunktion
TOcApp *702*
TRegistrar *771*
SetOuter Elementfunktion
TUnknown *774*
SetPageNumber
TPreviewPage Elementfunktion *423*
SetPageSize
TScroller Elementfunktion *470*
SetPaletteEntries
TPalette Elementfunktion *417*
SetPaletteEntry
TPalette Elementfunktion *417*
SetParent
TWindow Elementfunktion *562*
SetPasswordChar
TEdit Elementfunktion *250*
SetPixel
TDC Elementfunktion *200*
SetPolyFillMode
TDC Elementfunktion *200*
SetPos Elementfunktion
TOcPart *736*
SetPosition
TScrollBar Elementfunktion *466*
TSlider Elementfunktion *473*
SetPrinter
TPrinter Elementfunktion *441*
SetPrintParams
TPrintout Elementfunktion *444*
SetProp
TVbxControl Elementfunktion *518*, *519*, *520*
TWindow Elementfunktion *562*
SetProperty
TDocument Elementfunktion *233*
TFileDocument Elementfunktion *267*
TStorageDocument Elementfunktion *494*
TView Elementfunktion *528*
SetRange
TGauge Elementfunktion *300*
TScrollBar Elementfunktion *466*

TScroller Datenelement *470*
TSlider Elementfunktion *473*
SetReadOnly
TEdit Elementfunktion *250*
SetRect
TEdit Elementfunktion *250*
SetRectNP
TEdit Elementfunktion *250*
SetRectRgn
TRegion Elementfunktion *458*
SetRedraw
TWindow Elementfunktion *562*
SetResourceHandler
TModule Elementfunktion *372*
SetRGBColor
TChooseColorDialog Elementfunktion *122*
SetRGBMsgId
TChooseColorDialog Datenelement *122*
SetROP2
TDC Elementfunktion *200*
SetRuler
TSlider Elementfunktion *473*
SetSBarRange
Tscroller Elementfunktion *470*
SetScale
TOleWindow Elementfunktion, geschützte *406*
SetScale Elementfunktion
TOcScaleFactor *751*
SetScrollPos
TWindow Elementfunktion *562*
SetScrollRange
TWindow Elementfunktion *562*
SetSel
TListBox Elementfunktion *328*
SetSelection
TEdit Elementfunktion *251*
TOleWindow Elementfunktion, geschützte *406*
SetSelIndex
TComboBox Elementfunktion *142*
TListBox Elementfunktion *328*
SetSelIndexes
TListBox Elementfunktion *328*
SetSelItemRange
TListBox Elementfunktion *328*
SetSelString
TComboBox Elementfunktion *143*
TListBox Elementfunktion *328*
SetSelStrings
TListBox Elementfunktion *328*
SetShqdowStyle
TButtonGadget Elementfunktion *111*
SetShrinkWrap
TGadget Elementfunktion *288*
SetSize
TGadget Elementfunktion *288*
TMetaFilePict Elementfunktion *366*
SetSize Datenelement
TOcScaleFactor *750*
SetSize Elementfunktion
TOcPart *736*
SetSpacing
TStatusBar Elementfunktion *488*
SetState
TCheckBox Elementfunktion *119*
SetStorage
TOleDocument Elementfunktion, öffentliche *377*
TStorageDocument Elementfunktion *494*
SetStretchBltMode
TDC Elementfunktion *201*
SetStyle
TCheckBox Elementfunktion *119*
SetSymbol Elementfunktion
TAutoCommand *664*
TAutoIterator *671*
SetSysModalWindow
TWindow Elementfunktion *563*
SetSystemPaletteUse
TDC Elementfunktion *201*
SetTabStops
TEdit Elementfunktion *251*
TListBox Elementfunktion *329*
SetTemplate
TDocument Elementfunktion *233*
SetText
TButtonGadgetEnabler Elementfunktion *115*
TComboBox Elementfunktion *143*
TCommandEnabler Elementfunktion *150*
TMenuItemEnabler Elementfunktion *361*
TMessagebar Elementfunktion *362*
TStatic Elementfunktion *485*
TTextGadget Elementfunktion *497*
SetTextAlign
TDC Elementfunktion *201*
SetTextCharacterExtra
TDC Elementfunktion *201*
SetTextColor

TDC Elementfunktion *202*
TPrintPreviewDC Elementfunktion *448*
SetTextJustification
TDC Elementfunktion *202*
SetTimer
TWindow Elementfunktion *563*
SetTitle
TDocument Elementfunktion *234*
SetTopIndex
TListBox Elementfunktion *329*
SetTransferBuffer
TWindow Elementfunktion *563*
SetUnits
TScroller Elementfunktion *470*
Setup
TPrinter Elementfunktion *440*
SetupDC
TOleWindow Elementfunktion, geschützte *407*
SetupThumbRgn
TSlider Elementfunktion *476*
SetupWindow
TButton Elementfunktion *108*
TClipboardViewer Elementfunktion *134*
TComboBox Elementfunktion *143*
TCommonDialog Elementfunktion *152*
TDecoratedFrame Elementfunktion *211*
TDialog Elementfunktion *218*
TEdit Elementfunktion *254*
TEditFile Elementfunktion *257*
TEditSearch Elementfunktion *259*
TFrameWindow Elementfunktion *284*
TInputDialog Elementfunktion *315*
TOleFrame Elementfunktion, geschützte *384*
TOleWindow Elementfunktion, geschützte *407*
TScrollBar Elementfunktion *466*
TSlider Elementfunktion *477*
TWindow Elementfunktion *572*
SetupWindow Elementfunktion
TOcApp *702*
TOcView *761*
SetValidator
TEdit Elementfunktion *251*
SetValue
TGauge Elementfunktion *300*
SetValue Konstante
TAutoStack *684*
SetVBXProperty
TVbxControl Elementfunktion *520*
SetViewMenu
TView Elementfunktion *528*
SetViewportExt
TDC Elementfunktion *202*
TPrintPreviewDC Elementfunktion *448*
SetViewportOrg
TDC Elementfunktion *202*
TPrintPreviewDC Elementfunktion *449*
SetVisible Elementfunktion
TOcPart *736*
SetWindow
TScroller Elementfunktion *471*
SetWindowExt
TDC Elementfunktion *203*
TPrintPreviewDC Elementfunktion *449*
SetWindowFont
TWindow Elementfunktion *563*
SetWindowLong
TWindow Elementfunktion *563*
SetWindowOrg
TDC Elementfunktion *203*
SetWindowPlacement
TWindow Elementfunktion *564*
SetWindowPos
TWindow Elementfunktion *564*
SetWindowText
TWindow Elementfunktion *565*
SetWindowWord
TWindow Elementfunktion *565*
SetWinMainParams
TApplication Elementfunktion *89*
SetWordBreakProc
TEdit Elementfunktion *251*
SetWorldTransform
TDC Elementfunktion *203*
ShadowStyle
TButtonGadget Datenelement *111*
ShareViMsgId
TOpenSaveDialog Datenelement *409*
ShareViolation
TOpenSaveDialog Elementfunktion *410*
short far*() Elementfunktion
TAutoVal *691*
short() Elementfunktion
TAutoVal *691*
ShouldDelete
TCelArray Datenelement *117*
TDC Datenelement *208*
TGdiObject Datenelement *305*

TMenu Datenelement *354*
Show
TWindow Elementfunktion *565*
Show Elementfunktion
TOcPart *736*
TOctool barInfo *753*
ShowCaret
TWindow Elementfunktion *565*
ShowList
TComboBox Elementfunktion *143*
ShowOwnedPopups
TWindow Elementfunktion *565*
ShowScrollBar
TWindow Elementfunktion *565*
ShowWindow
TWindow Elementfunktion *566*
shrinkToClient
im Konstruktor von TFrameWindow *279*
shrinktoClient und Ereignis der
Benachrichtigung des übergeordneten
Fensters *283*
ShrinkWrapHeight
TGadget Datenelement *289*
TGadgetWindow Datenelement *297*
ShrinkWrapWidth
TGadget Datenelement *289*
TGadgetWindow Datenelement *297*
Shutdown Elementfunktion
TOcView *762*
TRegistrar *771*
ShutDownWindow
TWindow Elementfunktion *566*
shxxxx Dokument-Sharing-Aufzählung
(Konstanten) *82*
Size
TDib Elementfunktion *224*
TMetaFilePict Elementfunktion *366*
TRect Elementfunktion *814*
TUIHandle Elementfunktion *510*
SizeMax
TChooseFontDialog::TData Datenelement
127
SizeMin
TChooseFontDialog::TData Datenelement
127
SizeOfResource
TModule Elementfunktion *373*
Skalenpositionen
Gleitregler und *473*
SlideDC
TSlider Datenelement *477*
SlideThumb
TSlider Elementfunktion *477*
Sliding
TSlider Datenelement *478*
SlotThick
TSlider Datenelement *478*
Snap
TSlider Datenelement *478*
Snapping *477*
SnapPos
TSlider Elementfunktion *477*
sOptionSet Elementfunktion
TOcModule *725*
Spacing
TStatusBar Datenelement *489*
Spaltenformat, Editierfelder, Einstellen *251*
Speicher
Freigeben *255*
Speicherabteile *706*
Speichern
Verbunddokumente *706*
Speicherplatzreservierung
Editierfeld Fehlermeldung *253*
Sprachkennung *75*
SS_LEFT-Konstante *484*
StartDoc
TPrintDC Elementfunktion *433*
StartPage
TPrintDC Elementfunktion *434*
StartScan
TDib Elementfunktion *224*
StashContainerPopups
TOleFrame Elementfunktion, geschützte
385
StashCount
TOleFrame Datenelement, geschütztes *385*
StashedContainerPopups
TOleFrame Datenelement, geschütztes *385*
Stat Datenelement
TXOle *779*
State
TButtonGadget Datenelement *111*
StaticName
TEditView Elementfunktion *260*
TListView Elementfunktion *333*
TOleView Elementfunktion, öffentliche *389*
TWindowView Elementfunktion *575*
Statische Dialogelemente
Namen von Registrierungsklassen *485*

Ressourcen und Verknüpfung mit Objekten *484*
Status
TModule Datenelement *367*
TWindow Datenelement *532*
Status-Code
TXCompatibility-Konstruktor *577*
Statuszeilen *486, 487, 488, 489*
erstellen *487*
Rahmen *489*
Text einfügen in *488*
Step Elementfunktion
TAutoIterator *672*
StorageI
TStorageDocument Datenelement *494*
StorageI Elementfunktion
TOcSaveLoad *748*
Streaming
TFileDocument *266*
Streams
Konstruieren *78*
offen *267*
StretchBlt
TDC Elementfunktion *203*
StretchDIBits
TDC Elementfunktion *204*
string() Elementfunktion
TAutoVal *692*
String-Ressourcen
Lokalisieren *336, 337*
Strings
ID-Konstanten *68, 69*
Länge *484*
Sortieren *481, 482, 483*
Suche nach *496*
TComboBoxData Datenelement *146*
TListBoxData Datenelement *330*
TStringLookupValidator Datenelement *496*
Überprüfung der Gültigkeit von *495*
Vergleichen *270*
StrokeAndFillPath
TDC Elementfunktion *204*
StrokePath
TDC Elementfunktion *204*
Struktur
BITMAP *830*
BITMAPCOREHEADER *831*
BITMAPCOREINF *831*
BITMAPINFO *833*
BITMAPINFOHEADER *835*
COMPAREITEMSTRUCT *836*
DELETEITEMSTRUCT *837*
DEVMODE *838*
TAutoBool *660*
TAutoCurrency *665*
TAutoDate *665*
TAutoDouble *666*
TAutoFloat *669*
TAutoLong *673*
TAutoShort *681*
TAutoString *684, 685*
TAutoType *686*
TAutoVoid *693*
TOcDragDrop *712*
TOcMenuDescr *723*
TOcSaveLoad *748*
TOctool barInfo *752*
TOcViewPaint *763, 764*
WIN API *829ff*
Struktur TLocaleString *336, 337*
Strukturen
BANDINFOSTRUCT *98*
TBandInfo *98*
TWindowAttr *529*
Style
TChooseFontDialog::TData Datenelement *127*
TWindowAttr-Struktur *529*
SubclassWindowFunction
TWindow Elementfunktion *566*
Suche
mit oder ohne Beachtung der Groß- u. Kleinschreibung *250*
Suchen *258*
Suchen mit Beachtung der Groß- u. Kleinschreibung *250*
Suchen ohne Beachtung der Groß- u. Kleinschreibung *250*
Suchen-und-Ersetzen-Attribute *271, 272*
SuspendThrow
TApplication Elementfunktion *90*
Symbol Datenelement
TAutoCommand *665*
TAutoStack *684*
Symbole
Registrierung *648*
SyncFont
TPrintPreviewDC Elementfunktion *449*
SysColorChange
TBitmapGadget Elementfunktion *102*

TButtonGadget Elementfunktion *111*
TGadget Elementfunktion *288*
Systemcarets
erstellen *536*

T

T
TResponseTableEntry Elementfunktion *460*
TabbedTextOut
TDC Elementfunktion *204*
TActionFunc typedef *82*
TActionMemFunc typedef *83*
Tag
TDocument Datenelement *227*
TView Datenelement *526*
TAlign
TTextGadget Datenelement *497*
TAlign Typdefinition *499*
TAnyDispatcher typedef *83*
TAnyPMF typedef *83*
TAppDictionary Klasse *93*
Typdefinitionen *95*
TApplication
Vererbungsdiagramm *865*
TApplication Klasse *83*
Datenelemente *83, 90*
Destruktor *85*
Elementfunktionen *85, 91*
Konstruktoren *85*
Tastenkürzel
Ressourcen-IDs *66*
Tastenkürzeltabellen *529*
TAutoBase Klasse *659*
Destruktor *660*
TAutoBool Struktur *660*
ClassInfo Datenelement *660*
TAutoCommand Klasse *660, 661, 662, 663, 664*
Attr Datenelement *665*
ClearFlag Elementfunktion *661*
Destruktor *661*
Execute Elementfunktion *662*
Fail Elementfunktion *662*
GetSymbol Elementfunktion *662*
Invoke Elementfunktion *662*
IsPropSet Elementfunktion *662*
Konstruktor *661*
LookupError Elementfunktion *662*
Record Elementfunktion *663*
Report Elementfunktion *663*
Return Elementfunktion *663*
SetCommandHook Elementfunktion *663*
SetErrorMesgHook Elementfunktion *663*
SetFlag Elementfunktion *664*
SetSymbol Elementfunktion *664*
Symbol Datenelement *665*
TCommandHook Typdefinition *661*
TErrorMsgHook Typdefinition *661*
TestFlag Elementfunktion *664*
Undo Elementfunktion *664*
Validate Elementfunktion *664*
TAutoCurrency far*() Elementfunktion
TAutoVal *692*
TAutoCurrency Struktur *665*
ClassInfo Datenelement *665*
TAutoCurrency() Elementfunktion
TAutoVal *692*
TAutoDate far*() Elementfunktion
TAutoVal *692*
TAutoDate Struktur *665*
ClassInfo Datenelement *665*
Date Datenelement *666*
Konstruktor *666*
operator double Elementfunktion *666*
TAutoDate() Elementfunktion
TAutoVal *692*
TAutoDelete
TGdiObject Aufzählungstyp *303*
TAutoDouble Struktur *666*
ClassInfo Datenelement *666*
TAutoEnumerator *667, 668*
TAutoFactory *97*
TAutoFloat Struktur *669*
ClassInfo Datenelement *669*
TAutoIterator Klasse *669, 670, 671, 672*
Copy Elementfunktion *670*
GetSymbol Elementfunktion *670*
Init Elementfunktion *670*
IUnknown Elementfunktion *671*
Konstruktor *672*
Owner Datenelement *673*
Return Elementfunktion *671*
SetSymbol Elementfunktion *671*
Step Elementfunktion *672*
Test Elementfunktion *672*
TAutoLong Struktur *673*
ClassInfo Datenelement *673*
TAutoObject *673, 674, 675*

TAutoObjectByVal *675, 676*
TAutoObjectDelete *676, 677*
TAutoProxy Klasse *677, 678, 679, 680*
Bind Elementfunktion *678*
Destruktor *678*
IDispatch&() Elementfunktion *679*
IDispatch*() Elementfunktion *679*
Invoke Elementfunktion *680*
IsBound Elementfunktion *679*
Konstruktor *680*
Lookup Elementfunktion *679*
MustBeBound Elementfunktion *680*
SetLang Elementfunktion *680*
Unbind Elementfunktion *680*
TAutoShort Struktur *681*
ClassInfo Datenelement *681*
TAutoStack Klasse *681, 682, 683*
ArgCount Datenelement *683*
ArgSymbolCount Datenelement *683*
CurrentArg Datenelement *683*
Destruktor *682*
Konstruktor *682*
LangId Datenelement *683*
operator *683*
Owner Datenelement *683*
SetValue Konstante *684*
Symbol Datenelement *684*
TAutoString Struktur *684, 685*
char*() Elementfunktion *686*
ClassInfo Datenelement *686*
const char far*() Elementfunktion *686*
Destruktor *685*
int() Elementfunktion *685*
Konstruktoren *684*
operator = Elementfunktion *685*
TAutoType Struktur *686*
GetType Elementfunktion *687*
TAutoVal Klasse *687, 688, 689, 690, 691, 692*
Clear Elementfunktion *689*
Copy Elementfunktion *689*
double far*() Elementfunktion *689*
double() Elementfunktion *689*
far*() Elementfunktion *692*
float far*() Elementfunktion *690*
float() Elementfunktion *690*
GetDataType Elementfunktion *690*
IDispatch&() Elementfunktion *690*
IDispatch*() Elementfunktion *690*
int far*() Elementfunktion *690*
int() Elementfunktion *690*
IsRef Elementfunktion *691*
IUnknown&() Elementfunktion *691*
IUnknown*() Elementfunktion *691*
long far*() Elementfunktion *691*
long() Elementfunktion *691*
operator = Elementfunktion *688*
short far*() Elementfunktion *691*
short() Elementfunktion *691*
string() Elementfunktion *692*
TAutoCurrency far*() Elementfunktion *692*
TAutoCurrency() Elementfunktion *692*
TAutoDate far*() Elementfunktion *692*
TAutoDate() Elementfunktion *692*
TBool far*() Elementfunktion *689*
TBool() Elementfunktion *689*
TUString*() Elementfunktion *692*
unsigned long far*() Elementfunktion *693*
unsigned long() Elementfunktion *693*
TAutoVoid Struktur *693*
ClassInfo Datenelement *693*
TBandInfo
Struktur *98*
TBButtonGadget
Vererbungsdiagramm *868*
TBitmap
Elementfunktionen *100, 101, 102*
Konstruktoren *99*
Operator *101*
Vererbungsdiagramm *866*
TBitmap Elementfunktionen
BitsPixel *100*
Height *101*
Planes *101*
Width *101*
TBitMapGadget
Vererbungsdiagramm *866*
TBitmapGadget Klasse *102*
Destruktor *102*
Elementfunktionen *102*
Konstruktor *102*
TBitSet Klasse *103*
Elementfunktionen *103*
Konstruktoren *103*
TBIVbxLibrary *104*
TBIVbxLibrary Klasse
Destruktor *105*
Konstruktor *105*
TBool far*() Elementfunktion
TAutoVal *689*

TBool() Elementfunktion
TAutoVal *689*
TBorders
TGadget Struktur *285*
TBorderStyle
TGadget Aufzählungstyp *286*
TBrush *105*
Elementfunktionen *106*
Konstruktoren *105*
TButton
Vererbungsdiagramm *867*
TButton Klasse *106*
Antworttabelle *108*
Datenelemente *107*
Elementfunktionen, Geschützte *107*
Konstruktoren *107*
Makros *108*
TButtonGadget Klasse
Konstruktoren *109*
TButtonGadgetEnabler Klasse *114*
Konstruktor *114*
TCelArray Klasse *115*
Datenelemente, Geschützte *117*
Destruktoren *116*
Elementfunktionen, öffentlich *116*
Konstruktoren *115*
TCEnabled
TTinyCaption Datenelement *500*
TCharSet Klasse *117*
Elementfunktionen *118*
Konstruktoren *117*
TCheckBox
Vererbungsdiagramm *869*
TCheckBox Klasse *118*
Antworttabelle *121*
Datenelemente, öffentlich *118*
Elementfunktionen *119*, *120*
Konstruktoren *118*
TChooseColorDialog
Vererbungsdiagramm *870*
TChooseColorDialog Klasse *121*
Antworttabelle *123*
Datenelemente, Öffentliche *122*
Elementfunktionen *122*
Konstruktor *121*
Makro *123*
TChooseColorDialog::TData Klasse *123*
TChooseFontDialog
Vererbungsdiagramm *872*
TChooseFontDialog Klasse *124*
Antworttabelle *125*
Datenelemente, Geschützte *124*
Elementfunktionen, Geschützte *124*
Konstruktor *124*
Makro *125*
TChooseFontDialog::TData Klasse *125*
TChooseFontDialog::TData struct
Datenelemente *125*
TClientDC
Konstruktor *127*
TClipboard Klasse *127*
Datenelemente *129*, *133*
Destruktor *129*
Elementfunktionen, öffentlich *129*
Konstruktor *133*
Operator bool *131*
TClipboardViewer
Vererbungsdiagramm *873*
TClipboardViewer Klasse *133*
Antworttabelle *134*
Datenelemente, Geschützte *133*
Elementfunktionen, geschützt *134*
Konstruktor *133*, *134*
Makro *134*
TColor
Klasse *135*
Konstruktoren *136*
TColor Datenelemente
Black *135*
Gray *135*
LtBlue *135*
LtCyan *135*
LtGray *135*
LtGreen *135*
LtMagenta *135*
LtRed *136*
LtYellow *136*
Value *138*
White *136*
TColor Elementfunktionen
Blue *137*
Flags *137*
Green *137*
Index *138*
operator COLORREF() *138*
operator== *137*
PalIndex *138*
PalRelative *138*
Red *138*
Rgb *138*

TComboBox
Vererbungsdiagramm *874*
TComboBox Klasse *139*
Datenelemente, öffentlich *139*
Elementfunktionen *140, 143*
Konstruktoren *139*
TComboBoxData
Vererbungsdiagramm *876*
TComboBoxData Klasse *144*
Datenelemente, geschützt *145*
Elementfunktionen, öffentlich *144*
Destruktor *144*
Konstruktor *144*
TCommandEnabler *540*
TCommandEnabler Klasse *146*
Datenelemente *150*
Elementfunktionen *150*
Konstruktor *149*
TCommandHook Typdefinition
TAutoCommand *661*
TCommmonDialog
Vererbungsdiagramm *878*
TCommonDialog Klasse *150*
Antworttabelle *153*
Datenelemente, Geschützte *152*
Elementfunktionen *152*
Konstruktor *151*
TComponentFactory
TAutoFactory Operator *98*
TComponentFactory Typdefinition *693*
TCondFunc Typ *153*
TCondMemFunc typedef *153*
TControl
Vererbungsdiagramm *879*
TControl Klasse *156*
Antworttabelle *159*
Elementfunktionen *157, 158*
Konstruktoren *157*
TControlBar
Vererbungsdiagramm *880*
TControlBar Klasse *153, 154*
Konstruktor *154*
TControlGadget
Vererbungsdiagramm *881*
TControlGadget Klasse *155*
Antworttabellen *156*
Destruktor *155*
Konstruktoren *155*
TCreatedDC
Destruktor *159*
Elementfunktion *160*
Konstruktoren *159*
TCursor
Destruktor *161*
Konstruktoren *160*
TCursor Klasse
Öffentliche Elementfunktion *161*
td Transferfunktionskonstanten
TComboBox::Transfer und *143*
TDC
Destruktor *162*
Konstruktoren *162*
TDC Datenelemente
Handle *207*
OrgBrush *207*
OrgFont *207*
OrgPalette *207*
OrgPen *208*
OrgTextBrush *208*
ShouldDelete *208*
TDC Elementfunktionen
AngleArc *162*
Arc *163*
BeginPath *163*
BitBlt *163*
CheckValid *208*
Chord *164*
CloseFigure *164*
DPtoLP *165*
DrawFocusRect *165*
DrawIcon *165*
DrawText *165*
Ellipse *167*
EndPath *167*
EnumFontFamilies *167*
EnumFonts *168*
EnumMetaFile *168*
EnumObjects *168*
ExcludeClipRect *169*
ExcludeUpdateRgn *169*
ExtFloodFill *169*
ExtTextOut *170*
FillPath *171*
FillRect *171*
FillRgn *171*
FlattenPath *171*
FloodFill *171*
FrameRect *172*
FrameRgn *172*
GetAspectRatioFilter *172*

GetAttributeHDC *208*
GetBkColor *172*
GetBkMode *172*
GetBoundsRect *172*
GetBrushOrg *173*
GetCharABCWidths *173*
GetCharWidth *173*
GetClipBox *173*
GetClipRgn *174*
GetCurrentObject *174*
GetCurrentPosition *174*
GetDCOrg *174*
GetDeviceCaps *174*
GetDIBits *175*
GetFontData *175*
GetGlyphOutline *175*
GetHDC *208*
GetKerningPairs *176*
GetMapMode *176*
GetNearestColor *177*
GetOutlineTextMetrics *177*
GetPixel *178*
GetPolyFillMode *178*
GetROP2() *178*
GetStretchBltMode *178*
GetSystemPaletteEntries *178*
GetSystemPaletteUse *178*
GetTabbedTextExtent *178*
GetTextAlign *179*
GetTextCharacterExtra *180*
GetTextColor *181*
GetTextExtent *181*
GetTextFace *181*
GetTextMetrics *181*
GetViewportExt *181*
GetViewportOrg *182*
GetWindowExt *182*
GetWindowOrg *182*
GrayString *182*
Init *208*
IntersectClipRect *183*
InvertRect *184*
InvertRgn *184*
LineTo *184*
LPtoDP *184*
MaskBlt *185*
ModifyWorldTransform *186*
MoveTo *187*
OffsetClipRgn *187*
OffsetViewportOrg *187*
OffsetWindowOrg *187*
operator HDC() *183*
PaintRgn *187*
PatBlt *188*
PathToRegion *188*
Pie *188*
PlayMetaFile *189*
PlayMetaFileRecord *189*
PlgBlt *189*
PolyBezier *190*
PolyBezierTo *190*
PolyDraw *191*
Polygon *192*
Polyline *192*
PolylineTo *192*
PolyPolygon *192*
PolyPolyline *193*
PtVisible *193*
RealizePalette *193*
Rectangle *193*
RectVisible *193*
ResetDC *194*
RestoreBrush *194*
RestoreDC *194*
RestoreFont *194*
RestoreObjects *194*
RestorePalette *194*
RestorePen *194*
RestoreTextBrush *194*
RoundRect *195*
SaveDC *195*
ScaleViewportExt *195*
ScaleWindowExt *195*
ScrollDC *196*
SelectClipPath *196*
SelectClipRgn *196*
SelectObject *197*
SelectStockObject *197*
SetBkColor *197*
SetBkMode *197*
SetBoundsRect *198*
SetBrushOrg *198*
SetDIBits *198*
SetDIBitsToDevice *199*
SetMapMode *199*
SetMapperFlags *199*
SetMiterLimit *200*
SetPixel *200*
SetPolyFillMode *200*
SetROP2 *200*

SetStretchBltMode *201*
SetSystemPaletteUse *201*
SetTextAlign *201*
SetTextCharacterExtra *201*
SetTextColor *202*
SetTextJustification *202*
SetViewportExt *202*
SetViewportOrg *202*
SetWindowExt *203*
SetWindowOrg *203*
SetWorldTransform *203*
StretchBlt *203*
StretchDIBits *204*
StrokeAndFillPath *204*
StrokePath *204*
TabbedTextOut *204*
TextOut *205*
TextRect *206*
UpdateColors *206*
WidenPath *206*
TDC Klasse
Konstruktoren *207*
TDecoratedFrame
Vererbungsdiagramm *882*
TDecoratedFrame Elementfunktion
Insert *209*
PreProcessMsg *210*
TDecoratedFrame Elementfunktionen
SetClientWindow *210*
TDecoratedFrame Klasse *209*
Antworttabelle *211*
Datenelemente, geschützte *210*
Elementfunktionen *209*, *210*
Konstruktor *209*
Makro *211*
TDecoratedFrame Typdefinitionen
TLocation *209*
TDecoratedMDIFrame
Vererbungsdiagramm *883*
TDecoratedMDIFrame Klasse *211*
Elementfunktionen, geschützte *212*
Antworttabelle *212*
Konstruktor *211*
TDesktopDC
Konstruktor *212*
tdGetData
TRangeValidator-Aufzählung *455*
TTransferDirection-Aufzählung *513*
tdGetData Konstante *143*
tdGetData-Konstante *466*, *485*
TDialog
Vererbungsdiagramm *885*
TDialog Klasse *212*
Antworttabelle *218*
Datenelemente, öffentlich *213*
Destruktor *214*
Elementfunktionen *214*, *217*
Konstruktor *213*
TDialogAttr Struktur *218*
TDib
Datenelemente *225*
Destruktor *220*
Elementfunktionen *222*
Konstruktoren *220*
TDib Elementfunktionen
ChangeModeToPal *221*
ChangeModeToRGB *221*
FindColor *221*
FindIndex *221*
GetBits *221*
GetColor *222*
GetColors *222*
GetIndex *222*
GetIndices *222*
GetInfoHeader *222*
Height *222*
InfoFromHandle *226*
IsOk *222*
IsPM *223*
LoadFile *226*
LoadResource *226*
MapColor *223*
MapIndex *223*
MapUIColors *223*
numColors *223*
NumScans *223*
Operator == *223*
Operator BITMAPINFOHEADER() *221*
Operator HANDLE() *222*
Operator TRgbQuad() *224*
Read *226*
SetColor *224*
SetIndex *224*
Size *224*
StartScan *224*
ToClipboard *224*
Usage *224*
Width *224*
WriteFile *224*
TDib Klasse

Typdefinition *219*
TDib Typdefinition
Map *219*
TDibDC
Konstruktor *226*
TDocManager
TDocManager Klasse Konstruktor *236*
Vererbungsdiagramm *886*
TDocManager Klasse *235*
Antworttabelle *242*
Datenelemente *237*
Elementfunktionen *237*, *241*
Makro *242*
TDocProp
TDocument Typdefinition *228*
TDocTemplate
Vererbungsdiagramm *886*
TDocTemplate Klasse
Destruktor *799*
Dokumente erzeugen *795*
Elementfunktionen, öffentlich *795*
Konstruktor *798*
TDocTemplate-Klasse
Templates, Erzeugen *62*
TDocTemplateT Klasse *799*
Elementfunktionen *800*
TDocument
Vererbungsdiagramm *886*
TDocument Klasse *226*
Datenelemente *227*, *234*
Destruktor *228*
Elementfunktionen *228*, *234*
Konstruktor *228*
TDocument Typdefinition
TDocProp *228*
TFileDocProp *265*
TDocument und Schließen von Dateien *266*
TDocument-Klasse
Templates, Erzeugen *62*
TDocument::List Klasse *234*
Destruktor *234*
Konstruktor *234*
TDropInfo
Klasse *801*
Konstruktoren *801*
TDropInfo Elementfunktionen
DragFinish *801*
DragQueryFile *801*
DragQueryFileCount *802*
DragQueryFileNameLen *802*
DragQueryPoint *802*
Operator HDROP() *802*
tdSetData
TRangeValidator-Aufzählung *455*
TTransferDirection-Aufzählung *513*
tdSetData Konstante *143*
tdSetData-Konstante *485*
tdSizeData
TRangeValidator-Aufzählung *455*
TTransferDirection-Aufzählung *513*
tdSizeData Konstante *143*
tdSizeData-Konstante *466*, *485*
tdXXXX-Konstanten
TWindow::TransferData und *566*
TEdgeConstraint Struktur *242*
Elementfunktionen, öffentlich *243*
TEdgeOrSizeConstraint Struktur *244*
Elementfunktionen, öffentlich *244*
TEdit
Vererbungsdiagramm *887*
TEdit Klasse *245*
Antworttabelle *254*
Datenelement *252*
Elementfunktionen *246*, *252*
Konstruktoren *246*
TEditFile
Vererbungsdiagramm *889*
TEditFile Klasse *255*
Antworttabelle *257*
Datenelemente *255*
Destruktor *255*
Elementfunktionen *255*, *257*
Konstruktor *255*
TEditSearch
Vererbungsdiagramm *891*
TEditSearch Klasse *258*
Antworttabelle *259*
Datenelemente *258*
Elementfunktionen *258*
Konstruktor *258*
TEditView
Vererbungsdiagramm *892*
TEditView Klasse *259*
Datenelemente *261*
Destruktor *260*
Elementfunktionen *260*, *261*
Konstruktor *259*
TEllipse
TRegion Datenelement *455*
Templates

Erzeugen *60, 62*
TOleFactoryBase *379*
TermInstance
TApplication Elementfunktion *92*
TError Typdefinition
TXAuto *776*
TErrorMsgHook Typdefinition
TAutoCommand *661*
TErrorTypdefinition
TXObjComp *777*
Test
FirstThat-Funktion *541*
Test Elementfunktion
TAutoIterator *672*
Testen der Benutzereingabe *270*
TestFlag Elementfunktion
TAutoCommand *664*
TEventHandler Klasse *264*
TEventInfo Klasse *262*
Datenelemente, öffentlich *263*
TEventStatus enum *263*
Text
Ändern *249*
Änderungen *484*
Auswählen *249, 251*
Bearbeiten *245*
Clipboard und *246*
Einfügen *249, 250*
Ersetzen *271, 459*
Kopieren *252*
Länge *484*
Löschen *247*
Statische Dialogelemente *485*
Statuszeile *488*
Suchen *271*
Suchen nach *250, 258*
TTextGadget Elementfunktion *498*
Text-Gadgets *497, 498*
TextLen
TComboBox Datenelement *139*
TTextGadget Elementfunktion *498*
TextLenTStatic Datenelement *484*
TextOut
TDC Elementfunktion *205*
TextRect
TDC Elementfunktion *206*
TFileBuf und TFileDocument *265*
TFileDocProp
TFileDocument Typdefinition *265*
TFileDocument
Vererbungsdiagramm *893*
TFileDocument Klasse *265*
Destruktor *265*
Elementfunktionen *266*
Konstruktoren *265*
TFileOpenDialog
Vererbungsdiagramm *894*
TFileOpenDialog Klasse *268*
Elementfunktionen, öffentlich *269*
Konstruktor *268*
TFileSaveDialog
Vererbungsdiagramm *896*
TFileSaveDialog Klasse *269*
Elementfunktionen, öffentlich *269*
Konstruktor *269*
TFileStreamBase und TFileDocument *265*
TFilterValidator Klasse *269*
Datenelemente, geschützte *270*
Elementfunktionen, öffentlich *270*
Konstruktor *270*
TFindDialog
Vererbungsdiagramm *898*
TFindDialog Klasse *270*
Elementfunktionen, geschützte *271*
Konstruktor *271*
TFindReplaceDialog Klasse *271, 273*
Datenelemente, geschützte *272*
Destruktor *273*
Elementfunktionen *272*
Konstruktor *271, 273*
TFindReplaceDialog::TData Klasse *273*
Datenelemente *273*
Fehler *273*
TFloatingFrame
Vererbungsdiagramm *900*
TFloatingFrame Klasse *500*
Antworttabelle *276*
Elementfunktionen, geschützt *276*
Elementfunktionen, öffentlich *276*
Konstruktor *275*
Konstruktoren, öffentlich *275*
Makros *276*
TFont
Elementfunktionen *277*
Konstruktoren *276*
TFrameWindow Klasse *278*
Antworttabelle *284*
Datenelemente *279*
Datenelemente,geschützte *282*
Destruktor *279*

Elementfunktionen,geschützte *282*
Elementfunktionen,öffentlich *279*
Konstruktoren *278*, *279*
TGadget Klasse *284*, *285*, *286*, *287*, *288*, *289*, *290*
Aufzählungstypen *285*
Borders Datenelement *289*
BorderStyle Datenelement *289*
Bounds Datenelement *288*
Clip Datenelement *285*
CommandEnable Elementfunktion *286*
Destruktor *285*
Elementfunktionen, öffentliche *286*
geschützte Datenelemente *288*
geschützte Elementfunktionen *289*
GetBorders Elementfunktion *286*
GetBorderStyle Elementfunktion *286*
GetBounds Elementfunktion *287*
GetDesiredSize Elementfunktion *287*
GetEnabled Elementfunktion *287*
GetId Elementfunktion *287*
GetInnerRect Elementfunktion *289*
GetMargins Elementfunktion *287*
GetOuterSizes Elementfunktion *287*
Id Datenelement *289*
Inserted Elementfunktion *289*
Invalidate Elementfunktion *289*
InvalidateRect Elementfunktion *290*
Konstruktor *285*
LButtonDown Elementfunktion *290*
LButtonUp Elementfunktion *290*
Margins Datenelement *289*
MouseEnter Elementfunktion *290*
MouseLeave Elementfunktion *290*
MouseMove Elementfunktion *290*
NextGadget Elementfunktion *287*
öffentliche Datenelemente *285*
Paint Elementfunktion *290*
PaintBorder Elementfunktion *290*
PtIn Elementfunktion *291*
Removed Elementfunktion *291*
SetBorders Elementfunktion *287*
SetBorderStyle Elementfunktion *287*
SetBound Elementfunktion *288*
SetEnabled Elementfunktion *288*
SetMargins Elementfunktion *288*
SetShrinkWrap Elementfunktion *288*
SetSize Elementfunktion *288*
ShrinkWrapHeight Datenelement *289*
ShrinkWrapWidth Datenelement *289*
SysColorChange Elementfunktion *288*
TBorders Struktur *285*
TBorderStyle Aufzählungstyp *286*
TMargins Struktur *286*
TrackMouse Datenelement *289*
Update Elementfunktion *291*
WideAsPossible Datenelement *285*
Window Datenelement *289*
TGadgetKlasse
Strukturen *285*
TGadgetWindow
Vererbungsdiagramm *902*
TGadgetWindow Klasse *292*, *293*, *294*, *295*, *296*, *297*, *298*, *299*
Antworttabelle *299*
AtMouse Datenelement *296*
BkgndBrush Datenelement *296*
Capture Datenelement *296*
Create Elementfunktion *297*
Datenelemente, geschützte *296*
Direction Datenelement *296*
DirtyLayout Datenelement *296*
Elementfunktionen, geschützte *297*
Elementfunktionen, öffentliche *293*
EvLButtonDown Elementfunktion *297*
EvLButtonUp Elementfunktion *298*
EvMouseMove Elementfunktion *298*
EvSize Elementfunktion *298*
EvSysColorChange Elementfunktion *298*
FirstGadget Elementfunktion *293*
Font Datenelement *296*
FontHeight Datenelement *297*
GadgetChangedSize Elementfunktion *293*
GadgetFromPoint Elementfunktion *293*
GadgetReleaseCapture Elementfunktion *293*
Gadgets Datenelement *297*
GadgetSetCapture Elementfunktion *294*
GadgetWithId Elementfunktion *294*
GetDesiredSize Elementfunktion *298*
GetDirection Elementfunktion *294*
GetFont Elementfunktion *294*
GetFontHeight Elementfunktion *294*
GetHintMode Elementfunktion *294*
GetInnerRect Elementfunktion *298*
GetMargins Elementfunktion *299*
HintMode Datenelement *297*
IdleAction Elementfunktion *294*
Insert Elementfunktion *294*
Konstruktor *293*

LayoutSession Elementfunktion *295*
LayoutUnitsToPixels Elementfunktion *299*
Margins Datenelement *297*
NextGadget Elementfunktion *295*
NumGadgets Datenelement *297*
Paint Elementfunktion *299*
PaintGadgets Elementfunktion *299*
PositionGadget Elementfunktion *299*
Remove Elementfunktion *295*
SetDirection Elementfunktion *295*
SetHintCommand Elementfunktion *295*
SetHintMode Elementfunktion *296*
SetMargins Elementfunktion *296*
SetShrinkWrap *296*
ShrinkWrapHeight Datenelement *297*
ShrinkWrapWidth Datenelement *297*
TileGadgets Elementfunktion *299*
Typdefinition *293*
WideAsPossible *297*
TGadgetWindowFont Klasse *291*
Konstruktor *292*
TGauge
Vererbungsdiagramm *904*
TGauge Klasse *300*, *301*
Antworttabelle *302*
BarColor Datenelement *301*
Datenelemente, geschützte *301*
Elementfunktionen, geschützte *301*
Elementfunktionen, öffentliche *300*
EvEraseBkgnd Elementfunktion *301*
GetRange Elementfunktion *300*
GetValue Elementfunktion *300*
IsHorizontal Datenelement *301*
Konstruktor *300*
LedSpacing Datenelement *301*
LedThick Datenelement *301*
Margin Datenelement *301*
Max Datenelement *301*
Min Datenelement *301*
Paint Elementfunktion *301*
PaintBorder Elementfunktion *302*
SetColor Elementfunktion *300*
SetLed Elementfunktion *300*
SetRange Elementfunktion *300*
SetValue Elementfunktion *300*
Value Datenelement *301*
TGdiObject Klasse *302*, *303*, *304*, *305*, *306*
CheckValid Elementfunktion *305*
Destruktor *303*
Elementfunktionen, öffentliche *304*
GetObject Elementfunktion *304*
Handle Datenelement *305*
Konstruktoren *306*
Makros *306*
RefAdd Elementfunktion *304*
RefCount Elementfunktion *304*
RefDec Elementfunktion *304*
RefFind Elementfunktion *304*
RefInc Elementfunktion *305*
RefRemove Elementfunktion *305*
ShouldDelete Datenelement *305*
Typdefinition *303*
TGdiObject::TXGdi Klasse *307*
Clone Elementfunktion *308*
Konstruktor *308*
Msg Elementfunktion *308*
Throw Elementfunktion *308*
TGroupBox Klasse
GetClassName Elementfunktion *310*
NotifyParent Datenelement *309*
SelectionChanged Elementfunktion *310*
THatch8x8Brush Klasse *310*, *311*
Hatch11F1 *311*
Datenelemente, öffentliche *311*
Reconstruct Elementfunktion *312*
TheClipboard
TClipboard Datenelement *133*
ThisOpen
TStorageDocument Datenelement *494*
ThisUnknown Elementfunktion
TUnknown *775*
Thread-Verarbeitung *549*
Throw *819*
TGdiObject::TXGdi Elementfunktion *308*
TModule::TXInvalidModule
Elementfunkltion *374*
TXInvalidMainWindow öffentliche
Elementfunktion *93*
TXMenu Öffentliche Elementfunktion *577*
TXOwl Elementfunktion *580*
TXWindow Elementfunktion *576*
Throw Elementfunktion *819*
THSlider Klasse *479*
ThumbRect
TSlider Datenelement *478*
ThumbResId
TSlider Datenelement *478*
ThumbRgn
TSlider Datenelement *478*
TIC Klasse *312*

Konstruktor *312*
TicGap
TSlider Datenelement *478*
TIcon Klasse *313*
Elementfunktionen, öffentliche *313*
GetIconInfo Elementfunktion *313*
Konstruktoren *313*
Operator HICON Elementfunktion *314*
TileChildren
TMDIClient Elementfunktion *343*
TileGadgets
TGadgetWindow Elementfunktion *299*
TToolBox Elementfunktion *507*
Timer-Ereignis *553*
TInputDialog Klasse *314*
buffer Datenelement *314*
BufferSize Datenelement *314*
Datenelemente, öffentliche *314*
Elementfunktionen, geschützte *315*
Elementfunktionen, öffentliche *315*
Konstruktor *314*
prompt Datenelement *314*
SetupWindow Elementfunktion *315*
TransferData Elementfunktion *315*
TInputDialog-Klasse
Dialogelement-IDs *66*
TInStream Klasse *315*
Konstruktor *315*
TinyCaption Klasse
Konstruktoren *501*
Titel *92*, *157*
Dokument *528*
ermitteln *550*
Fenster *530*, *532*
Setzen *217*, *559*
Titelleisten
Aktivieren *502*
Ausmaße bestimmen *500*
erstellen *499*, *500*, *501*, *502*, *503*, *504*, *505*
Minimierung *503*
Rückgabe der Größe *504*
Rückgabe von Dimensionen *502*
Rückgabe von Maßen *503*
Schließsymbole *499*, *505*
zeichnen *504*
Titelleisten,schließen *500*
Title
TPrintout Datenelement *445*
TWindow Datenelement *532*
TLangId
typedef *787*
TLayoutConstraint Struktur *315*, *316*
Datenelemente, öffentliche *316*
MyEdge Datenelement *316*
OtherEdge Datenelement *316*
Relationship Datenelement *316*
RelWin Datenelement *316*
union Datenelement *317*
Units Datenelement *317*
TLayoutConstraint-Struktur
TWidthHeight-Aufzählung *528*
TLayoutMetrics Klasse *317*, *318*, *319*
Datenelemente, öffentliche *318*
Height Datenelement *318*
Konstruktor *318*
Width Datenelement *318*
X, Y Datenelement *318*
TLayoutWindow
Vererbungsdiagramm *905*
TLayoutWindow Klasse *320*, *321*, *322*
Antworttabelle *323*
Beispiele *320*
ClientSize Datenelement *323*
Elementfunktionen, geschützte *323*
Elementfunktionen, öffentliche *322*
EvSize Elementfunktion *323*
GetChildLayoutMetrics Elementfunktion *322*
Konstruktor *322*
Layout Elementfunktion *322*
RemoveChildLayoutMetrics Elementfunktion *322*
SetChildLayoutMetrics Elementfunktion *322*
TLayoutWindow Klasse, Datenelemente, geschützte *323*
TListBox
Vererbungsdiagramm *906*
TListBox Klasse *323*, *324*, *325*, *326*, *327*, *328*, *329*
AddString Elementfunktion *324*
ClearList Elementfunktion *324*
DeleteString Elementfunktion *324*
DirectoryList Elementfunktion *324*
FindExactString Elementfunktion *325*
FindString Elementfunktion *325*
GetCaretIndex Elementfunktion *325*
GetClassName Elementfunktion *329*
GetCount Elementfunktion *325*
GetHorizontalExtent Elementfunktion *325*

GetItemData Elementfunktion *325*
GetItemHeight Elementfunktion *325*
GetItemRect Elementfunktion *326*
GetSel Elementfunktion *326*
GetSelCount Elementfunktion *326*
GetSelIndex Elementfunktion *326*
GetSelIndexes Elementfunktion *326*
GetSelString Elementfunktion *326*
GetSelStrings Elementfunktion *326*
GetString Elementfunktion *327*
GetStringLen Elementfunktion *327*
GetTopIndex Elementfunktion *327*
InsertString Elementfunktion *327*
Konstruktor *323*
SetCaretIndex Elementfunktion *327*
SetColumnWidth Elementfunktion *327*
SetHorizontalExtent Elementfunktion *327*
SetItemData Elementfunktion *327*
SetItemHeight Elementfunktion *328*
SetItemRect Elementfunktion *328*
SetSel Elementfunktion *328*
SetSelIndex Elementfunktion *328*
SetSelIndexes Elementfunktion *328*
SetSelItemRange Elementfunktion *328*
SetSelString Elementfunktion *328*
SetSelStrings Elementfunktion *328*
SetTabStops Elementfunktion *329*
SetTopIndex Elementfunktion *329*
Transfer Elementfunktion *329*
TListBox Klasse, Elementfunktionen, geschützte *329*
TListBox Klasse, Elementfunktionen, öffentliche *324*
TListBoxData Struktur *330, 331*
AddString Elementfunktion *330*
AddStringItem Elementfunktion *331*
Clear Elementfunktion *331*
Datenelemente, öffentliche *330*
Elementfunktionen, öffentliche *330*
GetItemDatas Elementfunktion *331*
GetSelIndices Elementfunktion *331*
GetSelString Elementfunktion *331*
GetSelStringLength Elementfunktion *331*
GetStrings Elementfunktion *331*
ItemDatas Datenelement *330*
Konstruktor *330*
ResetSelections Elementfunktion *331*
SelCount Datenelement *330*
Select Elementfunktion *331*
SelectString Elementfunktion *331*
SelIndices Datenelement *330*
SelStrings Datenelement *330*
Strings Datenelement *330*
TListView
Vererbungsdiagramm *907*
TListView Klasse *332, 333, 334, 335*
Antworttabelle *336*
CanClose Elementfunktion *332*
CmEditAdd Elementfunktion *333*
CmEditClear Elementfunktion *333*
CmEditCopy Elementfunktion *334*
CmEditCut Elementfunktion *334*
CmEditDelete Elementfunktion *334*
CmEditItem Elementfunktion *334*
CmEditPaste Elementfunktion *334*
CmEditUndo Elementfunktion *334*
CmSelChange Elementfunktion *334*
Create Elementfunktion *332*
Datenelement, öffentliches *332*
Datenelemente, geschützte *333*
DirtyFlag Datenelement *332*
Elementfunktionen, geschützte *333*
Elementfunktionen, öffentliche *332*
EvGetDlgCode Elementfunktion *334*
GetViewName Elementfunktion *333*
GetWindow Elementfunktion *333*
Konstruktor *332*
LoadData Elementfunktion *335*
MaxWidth Datenelement *333*
Origin Datenelement *333*
SetDocTitle Elementfunktion *333*
SetExtent Elementfunktion *335*
StaticName Elementfunktion *333*
VnCommit Elementfunktion *335*
VnDocClosed Elementfunktion *335*
VnIsDirty Elementfunktion *335*
VnIsWindow Elementfunktion *335*
VnRevert Elementfunktion *335*
TLocaleId Typdefinition *694*
TLocaleString Struktur
Compare Elementfunktion *337*
Elementfunktionen, öffentliche *337*
GetSystemLangId Elementfunktion *337*
GetUserLangId Elementfunktion *337*
IsNativeLangId Elementfunktion *337*
operator = Elementfunktion *338*
operator const char* Elementfunktion *337*
Translate Elementfunktion *338*
TLocation
TDecoratedFrame Typdefinition *209*

TLookupValidator Klasse *338*
Elementfunktionen, öffentliche *338*
IsValid Elementfunktion *338*
Konstruktor *338*
Lookup Elementfunktion *339*
TMargins
TGadget Struktur *286*
TMDIChild Klasse *339, 340*
Antworttabelle *341*
DefWindowProc Elementfunktion *340*
Destroy Elementfunktion *340*
Elementfunktionen, geschützte *340*
Elementfunktionen, öffentliche *340*
EvMDIActivate Elementfunktion *341*
Konstruktoren *339*
PerformCreate Elementfunktion *341*
PreProcessMsg Elementfunktion *340*
ShowWindow Elementfunktion *340*
TMDIClient
Vererbungsdiagramm *908*
TMDIClient Klasse *341, 342, 343, 344*
Antworttabelle *345*
ArrangeIcons Elementfunktion *342*
CascadeChildren Elementfunktion *342*
ClientAttr Datenelement *342*
CloseChildren Elementfunktion *342*
CmArrangeIcons Elementfunktion *343*
CmCascadeChildren Elementfunktion *343*
CmChildActionEnable Elementfunktion *344*
CmCloseChildren Elementfunktion *344*
CmCreateChild Elementfunktion *344*
CmTileChildren Elementfunktion *344*
CmTileChildrenHoriz Elementfunktion *344*
Create Elementfunktion *342*
CreateChild Elementfunktion *342*
Datenelement, öffentliches *342*
Elementfunktionen, geschützte *343*
Elementfunktionen, öffentliche *342*
EvMDICreate Elementfunktion *344*
EvMDIDestroy Elementfunktion *344*
GetActiveMDIChild Elementfunktion *343*
GetClassName Elementfunktion *344*
InitChild Elementfunktion *343*
Konstruktor *341*
PreProcessMsgElementfunktion *343*
TileChildren Elementfunktion *343*
TMDIFrame
Vererbungsdiagramm *909*
TMDIFrame Klasse *345, 346, 347*
DefWindowProc Elementfunktion *347*
Elementfunktion, geschützte *347*
Elementfunktionen, öffentliche *346*
FindChildMenu Elementfunktion *346*
GetClientWindow Elementfunktion *347*
GetCommandTarget Elementfunktion *347*
Konstruktoren *346*
PerformCreate Elementfunktion *347*
SetMenu Elementfunktion *347*
TMeasurementUnits Aufzählungstyp *348*
TMemoryDC Klasse *348*
Datenelement, geschütztes *349*
Elementfunktionen, öffentliche *348*
Konstruktor *348*
OrgBitmap Datenelement *349*
RestoreBitmap Elementfunktion *348*
RestoreObjects Elementfunktion *348*
SelectObject Elementfunktion *348*
TMenu Klasse *349, 350, 351, 352, 353, 354*
AppendMenu Elementfunktion *350*
CheckMenuItem Elementfunktion *350*
CheckValid Elementfunktion *350*
Datenelemente, geschützte *354*
DeepCopy Elementfunktion *354*
DeleteMenu Elementfunktion *350*
DrawItem Elementfunktion *351*
Elementfunktionen, öffentliche *350*
EnableMenuItem Elementfunktion *351*
GetHandle Elementfunktion *351*
GetMenuCheckMarkDimensions Elementfunktion *351*
GetMenuItemCount Elementfunktion *351*
GetMenuItemID Elementfunktion *351*
GetMenuState Elementfunktion *351*
GetMenuString Elementfunktion *352*
GetSubMenu Elementfunktion *352*
Handle Datenelement *354*
InsertMenu Elementfunktion *352, 353*
IsOK Elementfunktion *353*
Konstruktoren *349*
MeasureItem Elementfunktion *353*
ModifyMenu Elementfunktion *353*
operator = Elementfunktion *353*
operator HMENU Elementfunktion *353*
operator uint Elementfunktion *354*
RemoveMenu Elementfunktion *354*
SetMenuItemBitmaps Elementfunktion *354*
ShouldDelete Datenelement *354*
TMenuDescr Klasse *355, 356, 357, 358, 359, 360*

ClearContainerGroupCount Elementfunktion *360*
ClearServerGroupCount Elementfunktion *360*
Datenelemente, geschützte *360*
Elementfunktionen, öffentliche *359*
ExtractGroups Elementfunktion *360*
GetGroupCount Elementfunktion *359*
GetHandle Elementfunktion *359*
GetId Elementfunktion *359*
GetModule Elementfunktion *359*
GroupCount Datenelement *360*
Id Datenelement *360*
Konstruktor *358*
Merge Elementfunktion *360*
Module Datenelement *360*
SetModule Elementfunktion *359*
Typdefinition *359*
TMenuItemEnabler Klasse *361*
Datenelemente, geschützte *361*
Elementfunktionen, öffentliche *361*
Enable Elementfunktion *361*
GetMenu Elementfunktion *361*
GetPosition Elementfunktion *361*
HMENU Datenelement *361*
Konstruktor *361*
Position Datenelement *361*
SetCheck Elementfunktion *362*
SetText Elementfunktion *361*
TMessageBar Klasse *362*
Datenelemente, geschützte *362*
Elementfunktionen, geschützte *363*
Elementfunktionen, öffentliche *362*
GetDesiredSize Elementfunktion *363*
GetInnerRect Elementfunktion *363*
HighlightLine Datenelement *362*
HintText Datenelement *362*
Konstruktor *362*
PaintGadgets Elementfunktion *363*
SetHintText Elementfunktion *362*
SetText Elementfunktion *362*
TMetaFileDC Klasse *363*
Close Elementfunktion *364*
Elementfunktion, öffentliche *364*
Konstruktoren *363*
TMetaFilePict Klasse *364*, *365*
CalcPlaySize Elementfunktion *365*
Datenelemente, geschützte *366*
Elementfunktionen, öffentliche *365*
Extent Datenelement *366*
GetMetaFileBits Elementfunktion *365*
GetMetaFileBitsEx Elementfunktion *365*
Height Elementfunktion *365*
Konstruktoren *364*
MappingMode Elementfunktion *365*
Mm Datenelement *366*
operator < Elementfunktion *365*
operator HMETAFILE() Elementfunktion *365*
PlayOnto Elementfunktion *365*
SetMappingMode Elementfunktion *366*
SetSize Elementfunktion *366*
Size Elementfunktion *366*
ToClipboard Elementfunktion *366*
Width Elementfunktion *366*
TModule Klasse *366*, *367*, *368*, *369*, *370*, *371*, *372*
AccessResource Elementfunktion *368*
AllocResource Elementfunktion *368*
CopyCursor Elementfunktion *368*
CopyIcon Elementfunktion *368*
Datenelemente, geschützte *373*
Datenelemente, öffentliche *367*
Elementfunktionen, öffentliche *368*
ErrorElementfunktion *368*
ExecDialog Elementfunktion *369*
FindResource Elementfunktion *369*
GetClassInfo Elementfunktion *369*
GetClientHandle Elementfunktion *369*
GetInstance Elementfunktion *370*
GetInstanceData Elementfunktion *370*
GetModuleFileName Elementfunktion *370*
GetModuleUsage Elementfunktion *370*
GetName Elementfunktion *370*
GetParentObject Elementfunktion *370*
GetProcAddress Elementfunktion *370*
HInstance Datenelement *373*
InitModule Elementfunktion *370*
IsLoaded Elementfunktion *370*
Konstruktor *366*
LoadAccelerators Elementfunktion *371*
LoadBitmap Elementfunktion *371*
LoadCursor Elementfunktion *371*
LoadIcon Elementfunktion *371*
LoadMenu Elementfunktion *371*
LoadResource Elementfunktion *371*
LoadString Elementfunktion *372*
LowMemory Elementfunktion *372*
lpCmdLine Datenelement *367*
MakeWindow Elementfunktion *372*

Module Datenelement *367*
Name Datenelement *373*
operator == Elementfunktion *372*
operator HINSTANCE Elementfunktion *372*
RestoreMemory Elementfunktion *372*
SetInstance Elementfunktion *372*
SetName Elementfunktion *372*
SetResourceHandler Elementfunktion *372*
SizeOfResource Elementfunktion *373*
Status Datenelement *367*
ValidWindow Elementfunktion *373*
TModule::TXInvalidModule Klasse *373*
Clone Elementfunktion *374*
Elementfunktionen, öffentliche *374*
Konstruktor *373*
Throw Elementfunktion *374*
TOcApp Klasse *695, 696, 697, 698, 699, 700, 701, 702, 703*
AddUserFormatName Elementfunktion *696*
Basisklasse *695*
Browse Clipboard Elementfunktion *697*
Browse Elementfunktion *696*
CanClose Elementfunktion *697*
Clip Elementfunktion *697*
Convert Elementfunktion *698*
Destruktor *704*
Drag Elementfunktion *698*
EnableEditMenu Elementfunktion *698*
EvActivate Elementfunktion *699*
EvResize Elementfunktion *699*
EvSetFocus Elementfunktion *699*
ForwardEvent Elementfunktion *704*
GetName Elementfunktion *699*
GetNameList Elementfunktion *700*
GetRegistrar() Elementfunktion *700*
IsOptionSet Elementfunktion *700*
Konstruktor *703*
Paste Elementfunktion *700*
RegisterClass Elementfunktion *701*
RegisterClasses Elementfunktion *701*
ReleaseObject Elementfunktion *701*
SetOption Elementfunktion *702*
SetupWindow Elementfunktion *702*
TOcMenuEnable Typdefinition *695*
TranslateAccel Elementfunktion *702*
UnregisterClass Elementfunktion *703*
UnregisterClasses Elementfunktion *703*
TOcAppMode Aufzählungstyp *705*
TOcAspect Aufzählungstyp *704*
TOcDialogHelp Aufzählungstyp *706*
TOcDocument
Konstruktoren *708*
TOcDocument Klasse *706, 707, 708, 709, 710, 711*
Close Elementfunktion *708*
Destruktor *708*
GetActiveView Elementfunktion *708*
GetName Elementfunktion *709*
GetParts Elementfunktion *709*
GetStorage Elementfunktion *709*
Konstruktoren *708*
LoadParts Elementfunktion *709*
RenameParts Elementfunktion *710*
SaveParts Elementfunktion *710*
SaveToFile Elementfunktion *710*
SetActiveView Elementfunktion *711*
SetName Elementfunktion *711*
SetStorage Elementfunktion *711*
TOcDragDrop Struktur *712*
InitInfo Elementfunktion *712*
Pos Elementfunktion *712*
Where Elementfunktion *712*
TOcDropAction Aufzählungstyp *713*
TOcFormatList Klasse *713, 714*
Add Elementfunktion *714*
Clear Elementfunktion *714*
Count Elementfunktion *714*
Destruktor *714*
Find Elementfunktion *715*
IsEmpty Elementfunktion *715*
Konstruktor *714*
TOcFormatListIter Klasse *715, 716*
Current Elementfunktion *716*
operator int Elementfunktion *716*
Restart Elementfunktion *716*
TOcFormatName Klasse *717*
Destruktor *717*
GetId Elementfunktion *717*
GetName Elementfunktion *718*
Konstruktoren *717*
TOcInitHow Aufzählungstyp *718*
TOcInitInfo Klasse *718, 719, 720, 721*
Cld Datenelement *720*
Container Datenelement *719*
Data Datenelement *720*
Handle Datenelement *721*
HIcon Datenelement *719*
How Datenelement *719*

Path Datenelement *720*
ReleaseDataObject Elementfunktion *722*
Storage Datenelement *720*
TOcInitInfo Konstruktor *721*
Where Datenelement *720*
TOcInitWhere Aufzählungstyp *722*
TOcInvalidate Aufzählungstyp *722*
ToClipboard
TBitmap Elementfunktion *101*
TDib Elementfunktion *224*
TMetaFilePict Elementfunktion *366*
TPalette Elementfunktion *417*
TOcMenuDescr Struktur *723*
HMenu Datenelement *724*
Width Datenelement *724*
TOcMenuEnable Typdefinition
TOcApp *695*
TOcModule Klasse *724, 725*
Destruktor *725*
GetRegistrar Elementfunktion *725*
IsOptionSet Elementfunktion *725*
Konstruktor *724*
OcApp Datenelement *726*
OcInit Elementfunktion *725*
TOcNameList Klasse *726, 727*
Add Elementfunktion *727*
Clear Elementfunktion *728*
Count Elementfunktion *728*
Destruktor *727*
Detach Elementfunktion *728*
Find Elementfunktion *728*
IsEmpty Elementfunktion *728*
Konstruktor *727*
operator *727*
TOcPart Klasse *728, 729, 730, 731, 732, 733, 734, 735, 736*
Activate Elementfunktion *730*
Basisklasse *728*
Clse Elementfunktion *730*
Delete Elementfunktion *730*
Destruktor *737*
Detach Elementfunktion *730*
DoVerb Elementfunktion *730*
Draw Elementfunktion *731*
EnumVerbs Elementfunktion *731*
GetName Elementfunktion *731*
GetNameLen Elementfunktion *731*
GetPos Elementfunktion *732*
GetRect Elementfunktion *732*
GetServerName Elementfunktion *732*
GetSize Elementfunktion *732*
IsActive Elementfunktion *733*
IsLink Elementfunktion *733*
IsVisible Elementfunktion *733*
Konstruktoren *729*
Load Elementfunktion *734*
Open Elementfunktion *734*
operator == Elementfunktion *729*
Rename Elementfunktion *734*
Save Elementfunktion *734*
Select Elementfunktion *735*
SetActive Elementfunktion *735*
SetHost Elementfunktion *735*
SetPos Elementfunktion *736*
SetSize Elementfunktion *736*
SetVisible Elementfunktion *736*
Show Elementfunktion *736*
UpdateRect Elementfunktion *737*
TOcPartCollection Klasse *737, 738*
Add Elementfunktion *738*
Clear Elementfunktion *738*
Count Elementfunktion *738*
Destruktor *737*
Detach Elementfunktion *738*
Find Elementfunktion *738*
IsEmpty Elementfunktion *738*
Konstruktor *737*
Locate Elementfunktion *739*
SelectAll Elementfunktion *739*
TOcPartCollectionIter Klasse *739, 740*
Current Elementfunktion *740*
Konstruktor *740*
operator ++ Elementfunktion *740*
operator int Elementfunktion *740*
Restart Elementfunktion *740*
TOcPartName Aufzählungstyp *741*
TOcRegistrar Klasse *741, 742, 743*
Basisklasse *741*
BOleComponentCreate Elementfunktion *742*
CanUnload Elementfunktion *744*
CreateOcApp Elementfunktion *743*
Destruktor *742*
GetAppDescriptor Elementfunktion *743*
GetFactory Elementfunktion *744*
Konstruktor *742*
LoadBOle Elementfunktion *744*
TOcRemView Klasse *744, 745, 746, 747*
Basisklasse *744*
Copy Elementfunktion *746*

EvClose Elementfunktion *746*
GetContainerTitle Elementfunktion *746*
GetInitialRect Elementfunktion *746*
Invalidate Elementfunktion *746*
IsOpenEditing Elementfunktion *747*
Konstruktor *745*
Load Elementfunktion *747*
Rename Elementfunktion *747*
Save Elementfunktion *747*
TOcSaveLoad Struktur *748*
Release Elementfunktion *748*
StorageI Elementfunktion *748*
TOcScaleFactor Klasse *748*, *749*, *750*
Destruktor *750*
GetScale Elementfunktion *751*
GetScaleFactor Elementfunktion *751*
IsZoomed Elementfunktion *751*
Konstruktoren *749*
operator = Elementfunktion *750*
PartSize Datenelement *750*
SetScale Elementfunktion *751*
SetSize Datenelement *750*
TOcScrollDir Aufzählungstyp *751*
TOctool barInfo Struktur *752*
HBottomTB Elementfunktion *752*
HFrame Elementfunktion *752*
HLeftTB Elementfunktion *753*
HRightTB Elementfunktion *753*
HTopTB Elementfunktion *753*
Show Elementfunktion *753*
TOcVerb Klasse *753*, *754*
CanDirty Datenelement *754*
Konstruktor *754*
TypeName Datenelement *754*
VerbIndex Datenelement *755*
VerbName Datenelement *755*
TOcView Klasse *755*, *756*, *757*, *758*, *759*, *760*, *761*, *762*
ActivatePart Elementfunktion *756*
ActivePart Datenelement *762*
Basisklasse *755*
BrowseClipboard Elementfunktion *756*
BrowseLinks Elementfunktion *756*
Copy Elementfunktion *757*
Destruktor *761*
EvActivate Elementfunktion *757*
EvClose Elementfunktion *757*
EvResize Elementfunktion *757*
EvSetFocus Elementfunktion *758*
Extent Datenelement *762*
FormatList Datenelement *762*
ForwardEvent Elementfunktion *761*
GetActivePart Elementfunktion *758*
GetOcDocument Elementfunktion *758*
GetOrigin Elementfunktion *758*
GetWindowRect Elementfunktion *759*
Init Elementfunktion *761*
InvalidatePart Elementfunktion *759*
Konstruktor *755*
Link Datenelement *762*
OcApp Datenelement *762*
OcDocument Datenelement *763*
Origin Datenelement *763*
Paste Elementfunktion *759*
RegisterClipFormats Elementfunktion *759*
ReleaseObject Elementfunktion *760*
Rename Elementfunktion *760*
ScrollWindow Elementfunktion *760*
SetLink Elementfunktion *760*
SetupWindow Elementfunktion *761*
Shutdown Elementfunktion *762*
Win Datenelement *763*
WinTitle Datenelement *763*
TOcViewPaint Struktur *763*, *764*
Aspect Datenelement *764*
Clip Datenelement *764*
DC Datenelement *764*
Part Datenelement *764*
Pos Datenelement *764*
Toggle
TCheckBox Elementfunktion *120*
ToggleModeIndicator
TStatusBar Elementfunktion *489*
TOleAllocator Klasse *765*
Alloc Elementfunktion *766*
Destruktor *765*
Free Elementfunktion *766*
Konstruktor *765*
Mem Datenelement *766*
TOleClientDC *374*
TOleClientDC Klasse *374*
Konstruktor *374*
TOleDocument *375*, *376*, *377*
TOleDocument Klasse *375*, *376*, *377*
CanClose, öffentliche Elementfunktion *375*
Commit, öffentliche Elementfunktion *376*
Destruktor *375*
GetNewStorage, öffentliche Elementfunktion *376*

GetOcApp, öffentliche Elementfunktion *376*
GetOcDoc, öffentliche Elementfunktion *376*
InitDoc, öffentliche Elementfunktion *376*
Konstruktor *375*
öffentliche Elementfunktionen *375*
Open, öffentliche Elementfunktion *376*
PathChanged, öffentliche Elementfunktion *377*
PreOpen, öffentliche Elementfunktion *377*
Read, öffentliche Elementfunktion *377*
ReleaseDoc, öffentliche Elementfunktion *377*
SetOcDoc, öffentliche Elementfunktion *377*
SetStorage, öffentliche Elementfunktion *377*
Write, öffentliche Elementfunktion *377*
TOleFactoryBase *378*, *379*
TOleFactoryBase Klasse *378*, *379*
CreateApp Template *379*
CreateObject Template *380*
DestroyApp Template *380*
TOleFactoryBase *379*
TOleFrame *38ff*
TOleFrame Klasse *380ff*
AddUserFormatName, öffentliche Elementfunktion *381*
Antworttabelle *385*
CanClose, geschützte Elementfunktion *382*
CleanupWindow, geschützte Elementfunktion *382*
Datenelemente, geschützte *385*
Destroy, geschützte Elementfunktion *382*
DestroyStashedPopups, geschützte Elementfunktion *382*
Elementfunktionen *381*
Elementfunktionen, geschützte *382*
EvActivateApp, geschützte Elementfunktion *382*
EvOcAppBorderSpaceReq, geschützte Elementfunktion *383*
EvOcAppBorderSpaceSet, geschützte Elementfunktion *383*
EvOcAppDialogHelp, geschützte Elementfunktion *383*
EvOcAppFrameRect, geschützte Elementfunktion *383*
EvOcAppInsMenus, geschützte Elementfunktion *383*
EvOcAppMenus, geschützte Elementfunktion *383*
EvOcAppProcessMsg, geschützte Elementfunktion *383*
EvOcAppRestoreUI, geschützte Elementfunktion *384*
EvOcAppShutdown, geschützte Elementfunktion *384*
EvOcAppStatusText, geschützte Elementfunktion *384*
EvOcEvent, geschützte Elementfunktion *384*
EvSize, geschützte Elementfunktion *384*
EvTimer, geschützte Elementfunktion *384*
GetOcApp, öffentliche Elementfunktion *381*
GetRemViewBucket, öffentliche Elementfunktion *382*
HoldMenu, geschütztes Datenelement *385*
Konstruktor *381*
OcApp, geschütztes Datenelement *385*
SetOcApp, öffentliche Elementfunktion *382*
SetupWindow, geschützte Elementfunktion *384*
StashContainerPopups, geschützte Elementfunktion *385*
StashCount, geschütztes Datenelement *385*
StashedContainerPopups, geschütztes Datenelement *385*
TOleMDIFrame *386*, *387*
TOleMDIFrame Klasse *386*, *387*
DefWindowProc, geschützte Elementfunktion *387*
Elementfunktionen, geschützte *387*
EvActivateApp, geschützte Elementfunktion *387*
EvOcAppInsMenus, geschützte Elementfunktion *387*
Konstruktor *387*
TOleView *388*, *389*, *390*, *391*
TOleView Klasse *388*, *389*, *390*, *391*
Antworttabelle *391*
CanClose, geschützte Elementfunktion *389*
CreateOCView, geschützte Elementfunktion *390*
Elementfunktionen, öffentliche *389*
Elementfunktionen,l geschützte *389*
EvOcViewAttachWindow, geschützte Elementfunktion *390*

EvOcViewClose, geschützte
Elementfunktion *390*
EvOcViewInsMenus, geschützte
Elementfunktion *390*
EvOcViewLoadPart, geschützte
Elementfunktion *390*
EvOcViewOpenDoc, geschützte
Elementfunktion *390*
EvOcViewPartInvalid, geschützte
Elementfunktion *391*
EvOcViewSavePart, geschützte
Elementfunktion *391*
GetViewName, öffentliche
Elementfunktion *389*
GetWindow, öffentliche
Elementfunktion *389*
Konstruktor *389*
SetDocTitle, öffentliche
Elementfunktion *389*
StaticName, öffentliche
Elementfunktion *389*
VnInvalidateRect, geschützte
Elementfunktion *391*
TOleWindow *392ff*
TOleWindow Klasse *392ff*
Antworttabelle *407*
CanClose, geschützte Elementfunktion *396*
CeEditConvert, geschützte
Elementfunktion *396*
CeEditCopy, geschützte
Elementfunktion *396*
CeEditCut, geschützte
Elementfunktion *396*
CeEditDelete, geschützte
Elementfunktion *396*
CeEditInsertObject, geschützte
Elementfunktion *397*
CeEditLinks, geschützte
Elementfunktion *397*
CeEditObject, geschützte
Elementfunktion *397*
CeEditPaste, geschützte
Elementfunktion *397*
CeEditPasteLink, geschützte
Elementfunktion *397*
CeEditPasteSpecial, geschützte
Elementfunktion *397*
CeEditVerbs, geschützte
Elementfunktion *397*
CleanupWindow, geschützte
Elementfunktion *398*
CmEditConvert, geschützte
Elementfunktion *398*
CmEditCopy, geschützte
Elementfunktion *398*
CmEditCut, geschützte
Elementfunktion *398*
CmEditDelete, geschützte
Elementfunktion *398*
CmEditInsertObject, geschützte
Elementfunktion *398*
CmEditLinks, geschützte
Elementfunktion *398*
CmEditPaste, geschützte
Elementfunktion *398*
CmEditPasteLink, geschützte
Elementfunktion *398*
CmEditPasteSpecial, geschützte
Elementfunktion *398*
CmFileClose, geschützte Element-
funktion *399*
CreateOcView, geschützte
Elementfunktion *399*
CreateVerbPopup, geschützte
Elementfunktion *399*
Datenelemente, geschützte *394*
Elementfunktioen, öffentliche *393*
Elementfunktionen, geschützte *396*
EvCommand, geschützte
Elementfunktion *399*
EvCommandEnable, geschützte
Elementfunktion *399*
EvDoVerb, geschützte
Elementfunktion *400*
EvLButtonDblClk, geschützte
Elementfunktion *400*
EvLButtonDown, geschützte
Elementfunktion *400*
EvLButtonUp, geschützte
Elementfunktion *400*
EvMDIActivate, geschützte
Elementfunktion *400*
EvMouseMove, geschützte
Elementfunktion *400*
EvOcEvent, geschützte
Elementfunktion *400*
EvOcPartInvalid, geschützte
Elementfunktion *401*

EvOcViewAttachWindow, geschützte Elementfunktion *401*
EvOcViewBorderSpaceReq, geschützte Elementfunktion *401*
EvOcViewBorderSpaceSet, geschützte Elementfunktion *401*
EvOcViewClipData, geschützte Elementfunktion *401*
EvOcViewClose, geschützte Elementfunktion *402*
EvOcViewDrag, geschützte Elementfunktion *402*
EvOcViewDrop, geschützte Elementfunktion *402*
EvOcViewGetPalette, geschützte Elementfunktion *402*
EvOcViewGetScale, geschützte Elementfunktion *402*
EvOcViewGetSiteRect, geschützte Elementfunktion *402*
EvOcViewInsMenus, geschützte Elementfunktion *402*
EvOcViewLoadPart, geschützte Elementfunktion *403*
EvOcViewOpenDoc, geschützte Elementfunktion *403*
EvOcViewPaint, geschützte Elementfunktion *403*
EvOcViewPartInvalid, geschützte Elementfunktion *403*
EvOcViewPartSize, geschützte Elementfunktion *403*
EvOcViewSavePart, geschützte Elementfunktion *403*
EvOcViewScroll, geschützte Elementfunktion *404*
EvOcViewSetScale, geschützte Elementfunktion *404*
EvOcViewSetSiteRect, geschützte Elementfunktion *404*
EvOcViewShowTools, geschützte Elementfunktion *404*
EvOcViewTitle, geschützte Elementfunktion *404*
EvRButtonDown, geschützte Elementfunktion *404*
EvSetCursor, geschützte Elementfunktion *405*
EvSetFocus, geschützte Elementfunktion *405*
EvSize, geschützte Elementfunktion *405*
GetInsertPosition, geschützte Elementfunktion *405*
GetLogPerUnit, geschützte Elementfunktion *405*
GetOcApp, öffentliche Elementfunktion *393*
GetOcDoc, öffentliche Elementfunktion *393*
GetOcRemView, öffentliche Elementfunktion *394*
GetOcView, öffentliche Elementfunktion *394*
HasActivePart, öffentliche Elementfunktion *394*
Init, geschützte Elementfunktion *405*
InvalidatePart, geschützte Elementfunktion *406*
Konstruktor *393*
Paint, geschützte Elementfunktion *406*
PaintMetafile, geschützes Datenelement *394*, *395*, *396*
PaintMetafile, geschützte Elementfunktion *397*
PaintMetafile, öffentliche Elementfunktion *394*
PaintParts, geschützte Elementfunktion *406*
Select, geschützte Elementfunktion *406*
SelectEmbedded, geschützte Elementfunktion *406*
SetScale, geschützte Elementfunktion *406*
SetSelection, geschützte Elementfunktion *406*
SetupDC, geschützte Elementfunktion *407*
SetupWindow, geschützte Elementfunktion *407*
VnInvalidateRect, geschützte Elementfunktion *407*
Toolbox-Gadgets *506*
TOpenSaveDialog Klasse *408*, *409*, *410*
CmLbSelChanged Elementfunkton *410*
CmOk Elementfunkton *410*
Data Datenelement *409*
Datenelemente, geschützte *409*
DialogFunction Elementfunkton *410*
DoExecute Elementfunkton *410*
Elementfunktionen, geschützte *410*
Elementfunktionen, öffentliche *409*
GetFileTitle Elementfunkton *409*

GetFileTitleLen Elementfunkton *409*
Init Elementfunkton *410*
Konstruktoren *409*
ofn Datenelement *409*
ShareViMsgId Datenelement *409*
ShareViolation Elementfunkton *410*
TOpenSaveDialog::TData Struktur *411*, *412*
CustomFilter Datenelement *411*
Datenlemente, öffentliche *411*
DefExt Datenelement *411*
Elementfunktionen, öffentliche *413*
Error Datenelement *411*
FileName Datenelement *412*
Filter Datenelement *412*
FilterIndex Datenelement *412*
Flags Datenelement *412*
InitialDir Datenelement *413*
SetFilter Elementfunktion *413*
TopLeft
TRect Elementfunktion *814*
TopRight
TRect Elementfunktion *814*
torage Datenelement
Siehe TOcInitInfo
Touches
TRect Elementfunktion *814*
TRegion Elementfunktion *459*
TOutStream Klasse *413*
Konstruktor *413*
TPaintDC Klasse *413*
Datenelemente, öffentliche *414*
Konstruktor *414*
Ps Datenelement *414*
TPalette Klasse *414*, *415*, *416*, *417*
AnimatePalette Elementfunktion *415*
Create Elementfunktion *417*
Elementfunktionen, geschützte *417*
Elementfunktionen, öffentliche *415*
GetNearestPaletteIndex Elementfunktion *415*
GetNumEntries Elementfunktion *416*
GetObject Elementfunktion *416*
GetPaletteEntries Elementfunktion *416*
GetPaletteEntry Elementfunktion *416*
Konstruktoren *414*
operator HPALETTE() Elementfunktion *416*
operator< Elementfunktion *416*
ResizePalette Elementfunktion *417*
SetPaletteEntries Elementfunktion *417*
SetPaletteEntry Elementfunktion *417*
ToClipboard Elementfunktion *417*
UnrealizeObject Elementfunktion *417*
TPaletteEntry Klasse *418*
Konstruktoren *418*
TPen Klasse *419*, *420*
Datenelemente, öffentliche *419*
Elementfunktionen, öffentliche *420*
GetObject Elementfunktion *420*
Konstruktoren *419*
operator HPEN() Elementfunktion *421*
TStockId Datenelement *419*
TPicResult Aufzählungstyp *421*
TPlacement Aufzählungstyp *421*
TPointer *805*, *806*
TPopupMenu Klasse *421*
Elementfunktionen, öffentliche *422*
Konstruktoren *421*
TrackPopupMenu Elementfunktion *422*
TPreviewPage Klasse *422*, *423*
Datenelemente, geschützte *423*
Elementfunktionen, geschützte *423*
Elementfunktionen, öffentliche *423*
EvSize Elementfunktion *423*
Konstruktor *422*
PageNum Datenelement *423*
Paint Elementfunktion *423*
PrintDC Datenelement *423*
PrintExtent Datenelement *423*
Printout Datenelement *423*
SetPageNumber Elementfunktion *423*
TPrintDC Klasse *424*, *425*, *426*, *427*, *428*, *429*, *430*, *431*, *432*, *433*
AbortDoc Elementfunktion *424*
BandInfo Elementfunktion *425*
Datenelement, geschütztes *434*
DeviceCapabilities Elementfunktion *425*
DOCINFO Datenelement *434*
Elementfunktionen, öffentliche *424*
EndDoc Elementfunktion *428*
EndPage Elementfunktion *428*
Escape Elementfunktion *428*
Konstruktoren *424*
NextBand Elementfunktion *432*
QueryAbort Elementfunktion *432*
QueryEscSupport Elementfunktion *433*
SetAbortProc Elementfunktion *433*
SetCopyCount Elementfunktion *433*
StartDoc Elementfunktion *433*
StartPage Elementfunktion *434*

TPrintDialog Klasse *437*, *438*
CmSetup Elementfunktion *439*
Data Datenelement *438*
Datenelemente, geschützte *438*
DialogFunction Elementfunktion *439*
DoExecute Elementfunktion *438*
Elementfunktionen, geschützte *439*
Elementfunktionen, öffentliche *438*
GetDefaultPrinter Elementfunktion *438*
Konstruktor *438*
pd Datenelement *438*
TPrintDialog::TData Struktur *434*, *435*, *436*
ClearDevMode Elementfunktion *436*
ClearDevNames Elementfunktion *436*
Copies Datenelement *435*, *436*
Datenelemente, öffentliche *435*
Elementfunktionen, öffentliche *436*
GetDeviceName Elementfunktion *437*
GetDevMode Elementfunktion *437*
GetDevNames Elementfunktion *437*
GetDriverName Elementfunktion *437*
GetOutputName Elementfunktion *437*
Lock Elementfunktion *437*
SetDevMode Elementfunktion *437*
SetDevNames Elementfunktion *437*
TransferDC Elementfunktion *437*
Unlock Elementfunktion *437*
TPrinter Klasse *439*, *440*
ClearDevice Elementfunktion *439*
CreateAbortWindow Elementfunktion *440*
Data Datenelement *440*
Datenelemente, geschützte *440*
Elementfunktionen, geschützte *440*
Elementfunktionen, öffentliche *439*
Error Datenelement *440*
ExecPrintDialog Elementfunktion *440*
GetDefaultPrinter Elementfunktion *441*
Konstruktor *439*
Print Elementfunktion *440*
ReportError Elementfunktion *440*
SetPrinter Elementfunktion *441*
Setup Elementfunktion *440*
TPrinter::TXPrinter Klasse
Konstruktor *441*
TPrinterAbortDlg Klasse *441*
Elementfunktionen, öffentliche *442*
EvCommand Elementfunktion *442*
Konstruktor *442*
TPrintout Klasse *442*, *443*, *444*
Banding Datenelement *445*
BeginDocument Elementfunktion *443*
BeginPrinting Elementfunktion *443*
Datenelemente, geschützte *445*
DC Datenelement *445*
Elementfunktionen, öffentliche *443*
EndDocument Elementfunktion *443*
EndPrinting Elementfunktion *443*
ForceAllBands Datenelement *445*
GetDialogInfo Elementfunktion *443*
GetTitle Elementfunktion *444*
HasPage Elementfunktion *444*
Konstruktor *442*
PageSize Datenelement *445*
PrintPage Elementfunktion *444*
SetPrintParams Elementfunktion *444*
Title Datenelement *445*
Typdefinition *444*
WantBanding Elementfunktion *444*
WantForceAllBands Elementfunktion *444*
TPrintPreviewDC Klasse *445*, *446*, *447*, *448*, *449*
CurrentPreviewFont Datenelement *449*
Datenelemente, geschützte *449*
Elementfunktionen, geschützte *449*
Elementfunktionen, öffentliche *446*
GetAttributeHDC Elementfunktion *449*
GetDeviceCaps Elementfunktion *446*
LPtoSDP Elementfunktion *446*
OffsetViewportOrg Elementfunktion *446*
PrnDC Datenelement *449*
PrnFont Datenelement *449*
ReOrg Elementfunktion *446*
ReScale Elementfunktion *447*
RestoreFont Elementfunktion *447*
ScaleViewportExt Elementfunktion *447*
ScaleWindowExt Elementfunktion *447*
SDPtoLP Elementfunktion *447*
SelectObject Elementfunktion *448*
SelectStockObject Elementfunktion *448*
SetBkColor Elementfunktion *448*
SetMapMode Elementfunktion *448*
SetTextColor Elementfunktion *448*
SetViewportExt Elementfunktion *448*
SetViewportOrg Elementfunktion *449*
SetWindowExt Elementfunktion *449*
SyncFont Elementfunktion *449*
TPrintPreviewDC Klasse Konstruktor *446*
TProfile Klasse *450*
Elementfunktionen, öffentliche *450*
GetInt Elementfunktion *450*

GetString Elementfunktion *450*
Konstruktor *450*
WriteInt Elementfunktion *450*
WriteString Elementfunktion *450*
TPrüfungsklasse *511, 512, 513*
TPXPictureValidator Klasse *451, 452*
Datenelement, geschütztes *452*
Elementfunktionen, öffentliche *451*
Error Elementfunktion *451*
IsValidInput Elementfunktion *451*
Konstruktor *451*
Pic Datenelement *452*
Picture Elementfunktion *452*
TrackMenuSelection
TDecoratedFrame Datenelement *210*
TrackMode
TScroller Datenelement *468*
TrackMouse
TGadget Datenelement *289*
TrackPopupMenu
TPopupMenu Elementfunktion *422*
TRadioButton
Vererbungsdiagramm *911*
TRadioButton Klasse *453*
Antworttabellen *454*
Konstruktor *453*
Konstruktor *453*
TRangeValidator Klasse *454*
Geschützte Datenelemente *455*
Konstruktor *454*
Öffentliche Elementfunktionen *454*
Transfer
TCheckBox Elementfunktion *120*
TComboBox Elementfunktion *143*
TEdit Elementfunktion *251*
TListBox Elementfunktion *329*
TRangeValidator Elementfunktion *454*
TScrollBar Elementfunktion *466*
TStatic Elementfunktion *485*
TValidator Elementfunktion *512*
TWindow Elementfunktion *566*
TransferBuffer
TWindow Datenelement *569*
TransferData
TInputDialog Elementfunktion *315*
TWindow Elementfunktion *566*
Transferdaten
Größe *566*
TransferDC
TPrintDialog::TData Elementfunktion *437*
TRansferfunktionen *507*
Transfermechanismen
Puffer *569*
Transfermechanismus
Oberflächenobjekte *539*
Transferpuffer *569*
Translate
TLocaleString Elementfunktion *338*
TranslateAccel Elementfunktion
TOcApp *702*
TRect *810, 813, 814*
Konstruktoren *808*
TRect Elementfunktionen
Area *809*
BottomLeft *809*
BottomRight *809*
Contains *809*
Height *809*
Inflated *810*
InflatedBy *810*
IsEmpty *810*
IsNull *810*
Normalize *810*
Normalized *810*
Offset *811*
OffsetBy *811*
Operator!= *812*
Operator TPoint* *813*
Operator& *812*
Operator&= *812*
Operator+ *811*
Operator+= *812*
Operator- *811*
Operator-= *812*
Operator== *812*
Operator| *812*
Operator|= *813*
Set *813*
SetEmpty *813*
Size *814*
TopLeft *814*
TopRight *814*
Touches *814*
Width *814*
TRect Klasse *807ff*
TRegion
Konstruktor *455*
TEllipse Datenelement *455*
TRegion Elementfunktionen
Contains *457*

GetRgnBox *457*
operator!= *457*
operator HRGN() *458*
operator&= *458*
operator+= *457*
operator-= *458*
operator= *457*
operator== *457*
operator^= *458*
operator|= *458*
SetRectRgn *458*
Touches *459*
TRegion Klasse *455, 456, 457, 458*
TRegistrar Klasse *766, 767, 768, 769, 770, 771*
AppDesc Datenelement *772*
CanUnload Elementfunktion *768*
CreateAutoObject Elementfunktion *768*
Destruktor *767*
GetFactory Elementfunktion *769*
GetOptions Elementfunktion *769*
IsOptionSet Elementfunktion *769*
Konstruktor *767, 772*
ProcessCmdLine Elementfunktion *770*
RegisterAppClass Elementfunktion *770*
ReleaseAutoApp Elementfunktion *770*
Run Elementfunktion *771*
SetOption Elementfunktion *771*
Shutdown Elementfunktion *771*
UnregisterAppClass Elementfunktion *772*
TRelationshipUnits-Aufzählung *459*
Trenner
Breite und Höhe *472*
TReplaceDialog Klasse *459*
Geschützte Elementfunktionen *459*
Konstruktoren *459*
TResId *814, 815*
Konstruktoren *815*
TResID Elementfunktionen
IsString *815*
TResID Friends
Operator *815*
Operator<operator< *815*
TResponseTableEntry Datenelement *460*
TResponseTableEntry Klasse *460*
TRgbQuad
Klasse *461*
Konstruktoren *461*
TRgbTriple
Klasse *461*
Konstruktoren *462*
try-Schlüsselwort
und Exceptions *579*
TScrollBar Klasse *463, 464, 465, 466*
Antworttabellen *467*
geschützte Elementfunktionen *466*
Konstruktoren *463*
Warnung *463*
TScrollBarData-Struktur *467*
TScroller
Konvertierungen *76*
TScroller Klasse *468, 469, 470*
Destruktor *469*
Konstruktor *469*
Öffentliche Datenelemente *468*
TSeparatorGadget Klasse *471*
TShadowStyle
TButtonGadget Typdefinition *109*
TSize
Klasse *816, 817*
Konstruktor *816*
TSize Elementfunktionen
Magnitude *817*
Operator+ *817*
TSlider
Vererbungsdiagramm *913*
TSlider Klasse *472ff*
Antworttabelle *478*
Destruktoren *473*
geschützte Elementfunktionen *473*
Konstruktoren *472*
Öffentliche Elementfunktionen *473*
TSortedStringArray Klasse *481, 482, 483*
Elementfunktionen *482*
TState
TButtonGadget Elementfunktion *110*
TButtonGadget Typdefinition *109*
TStatic Klasse *484, 485*
Antworttabelle *486*
Geschützte Elementfunktionen *485*
Konstruktoren *484*
Öffentliche Datenelemente *484*
TStatus
TStatus Datenelement *486*
TStatus Klasse
Öffentliche Datenelemente *486*
TStatusBar
TStatusBar Typdefinition *487*
Vererbungsdiagramm *915*
TStatusBar Klasse *486, 487, 488, 489*

Geschützte Datenelemente *489*
Konstruktor *487*
Öffentliche Elementfunktionen *488*
Typdefinition *487*
TStockId
TPen Datenelement *419*
TStorageDocument Klasse *490, 491, 492, 493*
Datenelemente *494*
Destruktor *491*
Elementfunktionen *491*
Konstruktor *491*
Typdefinitionen *490*
TStream Klasse *494*
Öffentliche Elementfunktionen *495*
ÖffentlicheR Destruktor *494*
TStringLookupValidator Klasse *495*
Destructor *496*
geschützte Datenelemente *496*
Konstruktor *496*
TStyle
TUIHandle Typdefinition *508*
TSystemMenu Klasse *496*
Konstruktor *497*
TTextGadget
Vererbungsdiagramm *916*
TTileDirection Typdefinition *499*
TTinyCaption Klasse *499, 500, 501, 502, 503, 504, 505*
Destruktoren *501*
Geschützte Datenelemente *500*
geschützte Elementfunktionen *501*
Makros *505*
TToolBox
Vererbungsdiagramm *917*
TTransferDirection-Aufzählungstyp *507*
TType
TButtonGadget Typdefinition *109*
TGdiObject Aufzählungstyp *303*
TUIHandle Klasse *507, 508, 509, 510*
Elementfunktionen *510*
Konstruktor *509*
Typdefinition *508*
TUnknown Klasse *772, 773, 774, 775*
Aggregate Elementfunktion *773*
Destruktor *775*
GetOuter Elementfunktion *773*
GetRefCount Elementfunktion *774*
Konstruktor *775*
operator IUnknown&() Elementfunktion *774*
operator IUnknown*() Elementfunktion *774*
Outer Datenelement *775*
QueryObject Elementfunktion *775*
SetOuter Elementfunktion *774*
ThisUnknown Elementfunktion *775*
TUString*() Elementfunktion
TAutoVal *692*
TValidator Klasse
Destructor *511*
Konstruktor *511*
TVbxControl Klasse *514, 515, 516, 517, 518, 519, 520*
Antworttabellen *520*
Destruktor *515*
geschützte Elementfunktionen *520*
Konstruktor *515*
Öffentliche Elementfunktionen *515*
TVbxEventHandler Klasse *521, 522, 523, 524*
Antworttabellen *525*
Geschützte Elementfunktionen *525*
TView Klasse *525, 526, 527*
Destruktor *526*
geschützte Datenelemente *528*
Geschützte Elementfunktionen *528*
Konstruktor *526*
Öffentliche Datenelemente *526*
Öffentliche Elementfunktionen *526*
TWhere
TUIHandle Typdefinition *509*
TWidthHeight-Aufzählung *528*
TWindow
Vererbungsdiagramm *920*
TWindow Klasse *530ff*
Antworttabelle *573*
Destruktoren *533*
Elementfunktionen, Definieren *153*
Flag-Konstanten *536*
geschützte Datenelemente *568*
geschützte Elementfunktionen *569*
geschützter Konstruktor *569*
Konstruktoren *532, 533*
Öffentliche Datenelemente *531*
Öffentliche Elementfunktionen *533*
TWindow::TXWindow Klasse *576*
TWindowAttr-Struktur *529, 531*
Eigenschaftskonstanten *529*

Konstanten für erweiterte Eigenschaften *529*
Öffentliche Datenelemente *529*
Tmenu Ressource-ID *529*
TWindowDC
Destruktoren *574*
Konstruktoren *574*
TWindowDC Datenelemente
Wnd *574*
TWindowDC Klasse *574*
TWindowView
Vererbungsdiagramm *919*
TWindowView Klasse *574, 575*
Destruktoren *574*
Konstruktoren *574*
Öffentliche Elementfunktionen *575*
TXAuto Klasse *776*
Basisklasse *776*
ErrorCode Datenelement *776*
Konstruktor *776*
TError Typdefinition *776*
TXBase *819*
TXBase Klasse *818, 819*
TXBase Klasse *819*
TXInvalidMainWindow
Exception *91*
TXInvalidMainWindow Konstruktor *93*
TXMenu
TXMenu Konstruktor *577*
TXObjComp Klasse *777*
Basisklasse *777*
ErrorCode Elementfunktion *777*
Konstruktor *777*
TErrorTypdefinition *777*
TXOle Klasse *778, 779*
Basisklasse *778*
Check Elementfunktion *779*
Destruktor *779*
Konstruktoren *778*
Stat Datenelement *779*
TXOutOfMemory
TXOutOfMemory Konstruktor *578*
TXRegistry Klasse *780*
Basisklasse *780*
Check Elementfunktion *780*
Key Elementfunktion *781*
Konstruktoren *780*
TXWindow
Konstruktoren *576*
TXWindow-Konstruktor *576*
Typdefinition
TComponentFactory *693*
TDib Klasse *219*
TLocaleId *694*
Typdefinitionen
TAppDictionary Klasse *95*
TGadgetWindow Klasse *293*
TGdiObject Klasse *303*
TMenuDescr Klasse *359*
TPrintout Klasse *444*
TSortedStringArray Klasse *481*
Type
TButtonGadget Datenelement *112*
typedef
COLORREF *836*
TLangId *787*
typehelp Registrierschlüssel *781*
TypeName Datenelement
TOcVerb *754*
Typographische Konventionen *28*
Typumwandlung
Zeiger *638*

U

U_Dispatch *594*
U_LPARAM_Dispatch *594*
U_POINT_Dispatch *594*
U_POINTER_Dispatch *594*
U_U_Dispatch *595*
U_U_U_U_Dispatch *595*
U_WPARAM_LPARAM_Dispatch *595*
Übereinstimmungsmaske für Dateien *240*
Übergeordnete Fenster *532*
Umwandlung
Pixel in Twips *524*
Twips in Pixel *524*
Unbind Elementfunktion
TAutoProxy *680*
Uncheck
TCheckBox Elementfunktion *120*
Uncondemn
TApplication Elementfunktion *90*
UNDER CLASSTYPE_CLASSTYPE-Makro *529*
Undo
TEdit Elementfunktion *251*
Undo Elementfunktion
TAutoCommand *664*

Ungültig Machen von Schaltern *113*
Ungültige Zeichen
numerische Werte *454*
Suche *496*
Suche nach *511*
ungültiges Zeichen
Prüfen auf *270*
Unhandled
TXOwl Elementfunktion *580*
TXWindow Elementfunktion *576*
union
TLayoutConstraint Datenelement *317*
Units
TLayoutConstraint Datenelement *317*
Unlock
TPrintDialog::TData Elementfunktion *437*
UnlockBuffer
TEdit Elementfunktion *251*
UnrealizeObject
TBrush Elementfunktion *106*
TPalette Elementfunktion *417*
UnRefTemplate
TDocManager Elementfunktion *241*
UnregisterAppClass Elementfunktion
TRegistrar *772*
UnregisterClass Elementfunktion
TOcApp *703*
UnregisterClasses Elementfunktion
TOcApp *703*
UnregisterHotKey
TWindow Elementfunktion *567*
UnsetOption
TValidator Elementfunktion *513*
unsigned long far*() Elementfunktion
TAutoVal *693*
unsigned long() Elementfunktion
TAutoVal *693*
Untergeordnete Fenster
Anzahl *555*
Dekorieren *211*
Erzeugen *214*
Iterator-Elementfunktionen *541, 542*
nächstes *554*
vorhergehendes *556*
Untergeordnete Liste
Fenster einlesen in eine *543*
Fenstererstellung und *532*
Objekte löschen *538*
Test-Iteration und *541*
untergeordnete Fenster und *536*
Untergeordnete Listen
Oberflächenobjekte und *545*
Objekte löschen *533*
Untergeordnetes Fenster
letztes *546*
Update
TControlGadget Elementfunktion *156*
TGadget Elementfunktion *291*
UpdateColors
TDC Elementfunktion *206*
UpdateRect Elementfunktion
TOcPart *737*
UpdateWindow
TWindow Elementfunktion *567*
UpperBound
TSortedStringArray Elementfunktion *483*
Usage
TDib Elementfunktion *224*
usage Registrierschlüssel *781*

V

v_Activate_Dispatch *595*
v_Dispatch *595*
v_LPARAM_Dispatch *595*
v_MdiActivate_Dispatch *596*
v_ParentNotify_Dispatch *596*
v_POINT_Dispatch *596*
v_POINTER_Dispatch *596*
v_U_Dispatch *596*
v_U_POINT_Dispatch *597*
v_U_U_Dispatch *597*
v_U_U_U_Dispatch *597*
v_U_U_W_Dispatch *597*
v_WPARAM_Dispatch *597*
v_WPARAM_LPARAM_Dispatch *598*
Valid
TValidator Elementfunkrion *513*
Validate
TWindow Elementfunktion *567*
Validate Elementfunktion
TAutoCommand *664*
ValidateRect
TWindow Elementfunktion *567*
ValidateRgn
TWindow Elementfunktion *567*
ValidationError
TEdit Elementfunktion *251*
Validator

TEdit Datenelement *252*
Validatoren
Stringsuche *495*
ValidChars
TFilterValidator Datenelement *270*
ValidWindow
TModule Elementfunktion *373*
Value
TColor Datenelement *138*
TGauge Datenelement *301*
VARIANT Datentyp *687*
VBEXEVENT-Struktur
L-Parameter *522*
VBX-Elemente
Ereignisse absenden *521*
VBXEVENT-Struktur
Control *522*
EventIndex *522*
EventName *522*
ID *522*
NumParams *522*
ParamList *522*
Window *522*
Verbinderobjekte *695*
VerbIndex Datenelement
TOcVerb *755*
verbn Registrierschlüssel *782*
VerbName Datenelement
TOcVerb *755*
verbnopt Registrierschlüssel *782*
Verbunddokument *755*
Verbunddokumente
Speichern und Laden *706*
Vererbte Elementfunktionen *54*
Vererbungsdiagramm
Beschreibung *53*
TApplication *865*
TBitMap *866*
TBitMapGadget *866*
TButton *867*
TButtonGadget *868*
TCheckBox *869*
TChooseColorDialog *870*
TChooseFontDialog *872*
TClipboardViewer *873*
TComboBox *874*
TComboBoxData *876*
TCommonDialog *878*
TControl *879*
TControlBar *880*
TControlGadget *881*
TDecoratedFrame *882*
TDecoratedMDIFrame *883*
TDialog *885*
TDocManager *886*
TDocTemplate *886*
TDocument *886*
TEdit *887*
TEditFile *889*
TEditSearch *891*
TEditView *892*
TFileDocument *893*
TFileOpenDialog *894*
TFileSaveDialog *896*
TFindDialog *898*
TFloatingFrame *900*
TGadgetWindow *902*
TGauge *904*
TLayoutWindow *905*
TListBox *906*
TListView *907*
TMDIClient *908*
TMDIFrame *909*
TRadioButton *911*
TSlider *913*
TStatusBar *915*
TTextGadget *916*
TToolBox *917*
TWindow *920*
TWindowView *919*
version Registrierschlüssel *783*
Versionsnummern
Zurückgeben *82*
Vertikale Gleitregler *480*
Verwalten der Ereignistabelle einer View *227*
Verzeichnisse
Pfad registrieren *647, 657*
View-Ereignisbotschaften *651*
Views
Erzeugen *62*
ohne Fenster *525*
Schließen *62*
views, ID-Konstanten *69*
VnCommit
TEditView Elementfunktion *261*
TListView Elementfunktion *335*
VnDocClosed
TEditView Elementfunktion *261*
TListView Elementfunktion *335*

VnInvalidateRect
TOleView Elementfunktion, geschützte *391*
TOleWindow Elementfunktion, geschützte *407*
VnIsDirty
TEditView Elementfunktion *261*
TListView Elementfunktion *335*
VnIsWindow
TEditView Elementfunktion *261*
TListView Elementfunktion *335*
VnRevert
TEditView Elementfunktion *262*
TListView Elementfunktion *335*
vnxxxx
View-Benachrichtigungskonstanten *581, 582*
void
Typdefinition TSortedStringArray *481*
VScroll
TScroller Elementfunktion *471*

W

W
TDib Datenelement *225*
TWindowAttr Struktur *530*
WaitingForSysCmd
TTinyCaption Datenelement *501*
WantBanding
TPrintout Elementfunktion *444*
WantForceAllBands
TPrintout Elementfunktion *444*
Warnton *253*
Werte
Einstellen *270*
Maximum/Minimum setzen *455*
Überprüfung *512*
Wertebereiche
prüfen *454*
wfAutoCreate-Konstante *536*
wfXxxx-Konstanten *536*
White
TColor Datenelement *136*
WideAsPossible
TGadget Datenelement *285*
WidenPath
TDC Elementfunktion *206*
Width
TBitmap Elementfunktion *101*
TDib Elementfunktion *224*
TLayoutMetrics Datenelement *318*
TMetaFilePict Elementfunktion *366*
TRect Elementfunktion *814*
WIN API
Gekapselte Funktionen *821, 823, 825, 827*
Struktur *829, 831, 833, 835, 837, 839, 841, 843, 845, 847, 849, 851, 853, 855, 857, 859, 861, 863*
Win Datenelement
TOcView *763*
WIN32 DLLs
Exportieren *79*
Importieren *79*
Window
TGadget Datenelement *289*
TScroller Datenelement *468*
TXWindow Datenelement *576*
WindowFromPoint
TWindow Elementfunktion *567*
WindowProc
TWindow Elementfunktion *567*
Windows
Aktionsschalter-Oberflächenelement *106*
CreateDialogParam Funktion *214*
CTL3D DLL Unterstützung *213*
DialogBoxParam Funktion *215*
Dialogelementbotschaft *217*
Dialogelemente, Oberflächenelemente *156*
Editierfeld als Oberflächenelement *245*
EndDialog Funktion *215*
Kombinationsfenster *139*
Oberflächenelement *536*
Registrierklassen-Attribute *570*
Registrierklassenname *570*
Registrierungs-Klassenname *254*
Registrierungsklassenname *108, 121*
Schaltfeld-Oberflächenelement *453*
vordefinierte Klasse *283*
Voreinstellung *217*
WinHelp
TWindow Elementfunktion *567*
WinTitle Datenelement
TOcView *763*
WM_CREATE Botschaft *529*
WM_INITDIALOG Botschaft
TDialogAttr und *219*
WM_OCEVENT Botschaft *783, 784*
WM_PAINT Botschaft *283, 555*
WM_TIMER Botschaften *553, 563*

WM_VBXFIREEVENT Botschaft *521*
Wnd
TWindowDC Datenelement *574*
Wortumbruch-Funktionen *251*
WParam
Parameter einer Dialogelementbotschaft *217*
Write
TEditFile Elementfunktion *257*
TOleDocument Elementfunktion, öffentliche *377*
WriteFile
TDib Elementfunktion *224*
WriteInt
TProfile Elementfunktion *450*
WriteString
TProfile Elementfunktion *450*
WS_ Fenstereigenschaftskonstanten *139, 157*
WS_-Fensterattributkonstanten *453*
WS_HSCROLL-Konstante *463*
WS_VSCROLL-Konstante *463*

X

X
TWindowAttr Struktur *530*
X, Y
TLayoutMetrics Datenelement *318*
XLine
TScroller Datenelement *468*
XPage
TScroller Datenelement *468*
XPos
TScroller Datenelement *469*
XRange
TScroller Datenelement *469*
XRangeValue
TScroller Elementfunktion *471*
xs
TApplication Typdefinitionen *84*
xs Exception-Status Bit-Flags *90*
xsAlloc *85*
xsBadCast *84*
xsBadTypeid *84*
XScrollValue
TScroller Elementfunktion *471*
xsMsg *84*
xsOwl *85*
xsUnknown *84*
XUnit
TScroller Datenelement *469*

Y

Y
TWindowAttr Struktur *530*
YLine
TScroller Datenelement *468*
YPage
TScroller Datenelement *468*
YPos
TScroller Datenelement *469*
YRange
TScroller Datenelement *469*
YRangeValue
TScroller Elementfunktion *471*
YScrollValue
TScroller Elementfunktion *471*
YUnit
TScroller Datenelement *469*

Z

Zeichen
Editierfeld und *245, 248, 254*
Formatieren in den Puffer *247*
geblätterte Zahl *250*
gültiges *270*
in den Puffer formatieren *246*
Suche nach ungültigen *454*
ungültig *270*
üngültige suchen *511*
Zeilenende *247*
Zeichenelemente *114*
Zeichnen
Dialogelemente, durch übergeordnetes Fenster gezeichnet *157, 158*
Fenster *283*
Zeiger
void *650*
Elementfunktionen *83*
Transferpuffer *143*
Typumwandlung *638*
zurückgeben *650*
Zeiger auf ein Fenster
zurückliefern *546*
Zeiger auf eine Elementfunktion *460*

und Botschaftszuteilungen *591*
Zeilenendezeichen *247*
Zeilenrückläufe *247*
Zeilenumbruch *247, 249*
Zeilenvorschübe *247*
Zeitereignis *563*
Zugriffs-Spezifizierer *29*
Zugriffsoperator
Windows-API-Aufrufe *29*
Zuteilen von Windows-Botschaften *591*
Zuteilungsfunktion
Beschreibung *591*
Zwischenablage
Aktivieren *131*
Fenster *130, 133, 134*
Formate registrieren *646, 653*
Kette von Ansichten *133*
Text und *246, 247, 249, 250*
Windows-API-Kapselung *127*

Borland

Copyright © 1994, Borland International, Inc. Alle Rechte Vorbehalten. Alle Produktnamen von Borland sind eingetragene Warenzeichen der Borland International Inc.
Deutsche Ausgabe © 1994 Borland GmbH, Monzastraße 4c, D-63225 Langen, Telefon 06103/979-0, Fax 06103/979-290
Zweigniederlassung Schweiz, Solothurnstraße 31, CH-3303 Jegenstorf
Hauptsitz: 100 Borland Way, Scotts Valley, CA 95066-3249, +1 (408) 431-1000
Niederlassungen in: Australien, Belgien, Brasilien, Chile, Deutschland, Dänemark, Frankreich, Großbritannien, Hongkong, Italien, Japan, Kanada, Korea, Lateinamerika, Malaysia, Niederlande, Neuseeland, Schweden, Singapur, Spanien und Taiwan.
• Part # BCP1245GExxxxx